

AD614578

PROGRAMMING IN NETWORKS AND GRAPHS

by
Ellis Johnson

OPERATIONS RESEARCH CENTER

INSTITUTE OF ENGINEERING RESEARCH

ORC 65-1
JANUARY 1965

COPY 1 OF 1 63-20

HARD COPY	\$. 3.00
MICROFICHE	\$. 0.75

PROCESSING COPY

UNIVERSITY OF CALIFORNIA - BERKELEY

PROGRAMMING IN NETWORKS AND GRAPHS

by

Ellis L. Johnson
Operations Research Center
University of California, Berkeley

January 1965

ORC 65-1 (■)

This research has been partially supported by the Office of Naval Research under Contract Nonr-222(83) and the National Science Foundation under Grant GP-2633 with the University of California. Reproduction in whole or in part is permitted for any purpose of the United States Government.

CONTENTS

PROGRAMMING IN NETWORKS AND GRAPHS

1.	Introduction.....	1
2.	Concepts from Graph Theory.....	2
3.	The Linear Programming Problem.....	5
4.	The Simplex Method for Network Flows.....	11
5.	Phase I.....	22
6.	The Primal-Dual Method for Network Flows.....	27
7.	Flows with Gains.....	30
8.	Linear Programming in an Undirected Graph.....	37
9.	Integer Programming in an Undirected Graph.....	46
10.	The Degree-Constrained Subgraph Problem.....	58
11.	References.....	60

1. Introduction

This paper treats a certain class of linear programs, the corresponding graphical interpretation, and will bring together the graphical and the algebraic approach. The first problem is the network flow problem. The graphical approach and labeling procedure are due to Ford and Fulkerson [7]. The earlier linear programming approach was given by Dantzig [2].

In linear programming the concept of a basic solution to a linear system of equations and inequalities is fundamental since if there is an optimal solution to a linear program, then there is an optimal basic solution. A basis of a matrix A is a matrix B consisting of a maximal set of linearly independent columns of A , and a basic solution to the linear system $Ax = b$, $0 \leq x \leq \alpha$, is a solution x^0 for which there is a basis B of A such that $x_j^0 = 0$ or $x_j^0 = \alpha_j$ unless A^j is a column of B .

The rank of a matrix A is defined to be the maximum number of linearly independent columns of A . It is a well-known result of linear algebra that the rank is also equal to the maximum number of linearly independent rows, and that for every set of linearly independent columns of A with fewer columns than the rank of A , other columns of A can be added to the set while preserving the property of linear independence until the set has as many columns as the rank of A . Thus, a matrix B of independent columns of A is

a basis of A if, and only if, B has as many columns as the rank of A , or equivalently if, and only if, every column of A can be written as a linear combination of the columns of B .

A square $m \times m$ matrix is defined to be singular if its rank is less than m and is non-singular if its rank is equal to m . Another well-known result from linear algebra is that a system of equations with a square, non-singular coefficient matrix has a unique solution.

2. Concepts from Graph Theory

Definitions:

Graph, Vertices, Edges: A graph G is a finite set V of vertices v_1, \dots, v_m and a finite set E of pairs of vertices, $e_k = (v_i, v_j)$, called edges. The edge $e_k = (v_i, v_j)$ is said to be incident to the vertices v_i and v_j .

Network, Undirected Graph, Arcs: The edges can be ordered pairs or unordered pairs, and the edge is correspondingly called directed or undirected. A directed graph, or network, is a graph with all of the edges directed. In a network the edges are called arcs, although the term edge still includes both the directed and undirected case. Examples of directed graphs are transportation networks and communication networks. In a transportation network the vertices are junctions, and the arcs are connections, such as roads and air routes, between junctions. An undirected connection, for example a two-way street, can be replaced by two directed edges.

Subgraph, Spanning Subgraph: A subgraph H of G is a graph whose vertex set \hat{V} and edge set \hat{E} are subsets of V and E , respectively.

A spanning subgraph H of G is a subgraph with the same vertex set as G .

Path, Simple Path, Cycle: A path in a graph G is a sequence of vertices and edges, $(v_1, e_1, v_2, e_2, \dots, v_{n-1}, e_{n-1}, v_n)$, such that e_i is incident to both v_i and v_{i+1} . The vertices v_1, v_n are called the ends of the path P , and the path is from v_1 to v_n . A simple path is a path with distinct vertices. A cycle is a simple path together with an edge from the beginning to the end of the path.

Connected, Component: A connected graph is a graph with at least one path between every pair of vertices, and a graph which is not connected clearly consists of connected components.

Tree, End, Forest, Spanning Forest: A tree is a connected graph with no cycles, and an end of a tree is a vertex touching only one edge of the tree. A forest is a graph consisting of one or more unconnected trees. A spanning tree of a graph G is a tree which is a spanning subgraph of G , and a spanning forest of G is a forest which is a spanning subgraph of G .

Note that all of the definitions from path to spanning forest do not depend on whether any edges are directed or undirected.

Lemma 1 If there is a path from v_1 to v_k , then there is a simple path from v_1 to v_k .

Proof: Let $v_1, e_1, v_2, \dots, v_{k-1}, e_k, v_k$ be a path from v_1 to v_k . Let v_i be the first vertex which is repeated, so that v_1, \dots, v_{i-1} are distinct and do not appear again in the path. Then suppose v_j is the last listing of vertex v_i in the path. Omit the segment e_i, v_{i+1}, \dots, e_j from the path to form a new path $v_1, e_1, v_2, \dots, v_i, e_{j+1},$

v_{j+1}, \dots, v_k .

If the above reduction of the path is repeated, eventually a single path will result.

Lemma 2 The following is an inductive characterization of trees: a tree is either a single vertex or is two disjoint trees joined by a single edge incident to one vertex of one tree and one vertex of the other tree.

Proof. Clearly, a graph constructed in such a way is a tree. The harder part of the proof is to show that every tree satisfies the condition. If a tree T has no edge, then it is a single vertex.

If T has an edge, say $e_1 = (v_1, v_2)$, then there is no simple path between v_1 and v_2 not using e_1 because if there were, T would contain a cycle. Hence, by Lemma 1 there is no path from v_1 to v_2 not using e_1 . So if e_1 is removed from T , the remaining graph has at least two connected components T_1 and T_2 with v_1 in T_1 and v_2 in T_2 . From every vertex w in T there are simple paths to v_1 and v_2 ; therefore, there is a simple path to v_1 to v_2 not containing e_1 . Hence, removal of e_1 causes the remaining graph to have exactly two connected components T_1 and T_2 . They are trees because if either had a cycle, so would T .

Lemma 3 Every tree has at least one end, and if it has an edge, then it has at least two ends.

Proof: The proof is most easily done using lemma 2. Lemma 3 is true for a single vertex. Suppose it is true for two trees. Then adding an edge incident to one vertex of each tree will always leave one end in each tree. Hence the new tree will have at least two ends, and the lemma is proven.

Lemma 4 A tree with m vertices has $m-1$ edges.

Proof: The proof is immediate using lemma 2 and induction as in the proof of lemma 3.

Lemma 5 Every connected graph G contains a spanning tree T .

Proof: Define a subgraph T having the same vertex set as G and edge set chosen as follows. Initially, choose any edge of G to be in T . Thereafter, choose any edge of G that does not produce a cycle in T . When every edge in $G - T$ produces a cycle if added to T , then T is easily seen to be a spanning tree of G .

3. The Linear Programming Problem

For a network G , the vertex-arc incidence matrix is defined by

$$a_{ij} = \begin{cases} 0 & \text{if arc } e_j \text{ is not incident to vertex } v_i \\ 1 & \text{if arc } e_j = (v_i, v) \text{ for some } v \in V \\ -1 & \text{if arc } e_j = (v, v_i) \text{ for some } v \in V. \end{cases}$$

The linear programming problem is:

$$\begin{aligned} \text{minimize } z \text{ subject to } & Ax + Us = b, \quad 0 \leq x \leq \alpha, \quad 0 \leq s \leq \sigma, \\ & cx + cs = z \end{aligned}$$

where A is an $m \times n$ vertex-arc incidence matrix of a network G , and U is an $m \times \bar{n}$ matrix such that each column of U has one non-zero entry which is a $+1$ or a -1 . If a column of U has a $+1$ in row i , denote the variable by s_i^+ , and if a column of U has a -1 in row i , denote the variable by s_i^- . Let σ be denoted correspondingly σ_i^+ or σ_i^- , and denote the corresponding cost c by c_i^+ or c_i^- . The s_i^+ and s_i^- are called slack variables or slacks.

If a variable x_k was permitted to be negative, then it could be

replaced by two variables, its positive and negative parts, and an arc would be adjoined to G in the reverse direction of e_k . If a variable x_k had a lower bound $\beta > 0$ and $e_k = (v_i, v_j)$, then b_i could be replaced by $b_i - \beta$, b_j replaced by $b_j + \beta$, α_k replaced by $\alpha_k - \beta$, and the lower bound replaced by 0. Nothing is assumed about b_i being positive, negative, or zero. Hence, $0 \leq x \leq \alpha$, $\alpha > 0$, is completely general and includes lower bounds and unrestricted variables. Symbolically, $\alpha_k = +\infty$ means no upper bound is placed on x_k .

Similarly, $0 \leq s \leq \sigma$, $\sigma > 0$ is perfectly general. A slack must have an upper or lower bound in order to mean anything. A bound can be adjusted to zero as above, and then a negative slack s_i^+ can be replaced by a non-negative s_i^- , and visa versa. Note that only one of s_i^+ , s_i^- could be present at a given vertex v_i . No upper bound on a slack is symbolically represented by $\sigma_k = +\infty$.

The k^{th} column of A corresponds to arc e_k of the network, and a column of U with non-zero entry in row i corresponds to vertex v_i . The variable x_k can be thought of as a flow in arc e_k , the variable s_i^+ can be thought of as exogenous flow out of vertex v_i , and s_i^- can be thought of as flow into vertex v_i . The constraints, then, require that the net flow in vertex v_i be b_i .

Let A^0 denote the matrix $[A, U]$. For a matrix B of columns of A^0 , let F_B be the subgraph of G consisting of the vertices corresponding to columns of U and edges corresponding to columns of A together with vertices incident to such edges.

Theorem 1 If B is a basis of A^0 , then F_B is a spanning forest

of G .

Proof: If F_B is not a spanning subgraph of G , then some vertex of G , say vertex v_1 , is not in F_B . Then, every entry in row 1 of B is a zero. But some column of A^0 has a non-zero entry in row 1, and such a column cannot be written as a linear combination of columns of B , contradicting B being a basis.

The remainder of the proof consist of showing that F_B has no cycles. Suppose F_B has a cycle $v_1, e_1, v_2, \dots, v_k, e_k, v_{k+1} = v_1$. Then there are k column of B , say B^1, \dots, B^k , corresponding to e_1, \dots, e_k .

$$\text{Let } y_j = \begin{cases} 1 & \text{if } e_j = (v_j, v_{j+1}) \\ -1 & \text{if } e_j = (v_{j+1}, v_j) \end{cases}.$$

For v_1 not in the cycle, $\sum_{j=1}^k b_{1j} y_j = 0$ because none of the arcs

e_1, \dots, e_k are incident to v_1 so all $b_{1j} = 0$ for $j = 1, \dots, k$.

For v_1 in the cycle, there are four cases to consider:

$$(i) \quad e_1 = (v_{i-1}, v_i), e_{i+1} = (v_i, v_{i+1})$$

$$(ii) \quad e_1 = (v_i, v_{i-1}), e_{i+1} = (v_i, v_{i+1})$$

$$(iii) \quad e_1 = (v_{i-1}, v_i), e_{i+1} = (v_{i+1}, v_i)$$

$$(iv) \quad e_1 = (v_i, v_{i-1}), e_{i+1} = (v_{i+1}, v_i).$$

$$\text{For case (i), } b_{1j} = \begin{cases} 1 & j=1 \\ -1 & j=i+1 \end{cases} \text{ and } b_{1j} = 0 \text{ for } j \neq 1 \text{ or } i+1,$$

and $y_1 = 1, y_{i+1} = 1$ so $\sum_{j=1}^k b_{1j} y_j = 0$. The other three cases are

similar, and in all of them $\sum_{j=1}^k y_j B^j = 0$ contradicting B

having linearly independent columns. Hence, the theorem is proven.

The above proof can be thought of as picking a direction around the cycle and sending a flow of 1 around in that direction. A flow of -1 can be thought of as reversing the direction of the arc and then sending a flow of +1.

A vertex v_1 corresponding to a column of U in B , a basis of A^0 , will be called a root of the tree in F_B .

Theorem 2 For B a basis of A^0 , every tree of the forest F_B has at most one root.

Proof: Suppose some tree had two roots v_1 and v_k . A tree is connected so there is a simple path from v_1 to v_k in the tree, say $v_1, e_1, v_2, \dots, v_{k-1}, e_k, v_k$. Hence, there are $k+1$ columns of B , say B^1, \dots, B^{k+2} , corresponding to $e_1, e_2, \dots, e_k, v_1, v_k$, respectively. Let

$$y_{k+1} = \begin{cases} -1 & \text{if } B^{k+1} \text{ has a } -1 \text{ non-zero entry} \\ +1 & \text{if } B^{k+1} \text{ has a } +1 \text{ non-zero entry,} \end{cases}$$

$$y_{k+2} = \begin{cases} +1 & \text{if } B^{k+2} \text{ has a } +1 \text{ non-zero entry} \\ -1 & \text{if } B^{k+2} \text{ has a } -1 \text{ non-zero entry,} \end{cases}$$

$$y_j = \begin{cases} +1 & \text{if } e_j = (v_j, v_{j+1}) \\ -1 & \text{if } e_j = (v_{j+1}, v_j) \end{cases}, \quad j = 1, \dots, k.$$

Then, as in theorem 1, $\sum_{j=1}^{k+2} y_j B^j = 0$, contradicting linear independence

of the columns of B . Hence, the theorem is true.

The construction above, as before, can be thought of as sending a unit of flow into v_1 and out of v_k .

A tree with one root is called a rooted tree, and a forest with each tree having one root is called a rooted forest.

A non-singular, triangular matrix is a square matrix with non-zeros on the main diagonal and all zeros below the main diagonal, or which can be brought to such a form by swapping rows and swapping columns. An equivalent, inductive characterization is the following: a square matrix B is non-singular, triangular if there is a row of B with only one non-zero entry and if the matrix \bar{B} formed from B by deleting that row and the column containing the non-zero entry is also non-singular, triangular. The above characterization is complete if a 1×1 non-zero matrix is understood to be non-singular and triangular.

Theorem 3 If F_B is a rooted, spanning forest of G , then B is a non-singular, triangular matrix.

Proof: Such a matrix B will be square by lemma 4, which says a tree has one less edge than vertex. The additional column of U for each tree makes B have as many columns as rows.

The proof is by induction on m , the number of rows of A^0 . For $m = 1$, B is a 1×1 non-zero matrix which is non-singular and triangular. Assume the theorem is true for $1, \dots$, or $m-1$ rows in A^0 for some $m \geq 2$. Consider a matrix A^0 having m rows.

If B has only columns from U , then B is diagonal so certainly non-singular and triangular. If B has a column from A , then F_B has an edge, and the tree to which the edge belongs has at least two ends by lemma 3. But the tree has only one root, and hence, there must be a vertex v_1 which is an end of the tree and not a root. Then, row 1 of B has only one non-zero entry. Let \bar{B}

be the matrix formed from B by deleting row i and the column with non-zero entry in row i . Let $\bar{A}^0 = [\bar{A}, \bar{U}]$ denote the matrix formed from A^0 by deleting row i and all columns with non-zero entries in row i , and let \bar{G} denote the network formed from G by deleting vertex v_i and all arcs incident to vertex v_i . Then \bar{A} is the vertex-arc incidence matrix of \bar{G} , and $F_{\bar{B}}$ is a spanning forest of \bar{G} . Furthermore, every tree of $F_{\bar{B}}$ has exactly one vertex corresponding to \bar{U} in \bar{B} . Hence, by the induction hypothesis, \bar{B} is non-singular and triangular. Therefore, B is non-singular and triangular, completing the proof.

Theorem 4 Let A^0 be such that every connected component of the network G has at least one vertex corresponding to a column of U . Then a matrix B of columns of A^0 is a basis if, and only if, F_B is a rooted, spanning forest of G .

Proof: By Lemma 5, each connected component of G contains a spanning tree. Let \hat{B} consist of the columns of A^0 corresponding to all the edges in the spanning trees of the connected components of G together with one column from U for each connected component of G . Then by theorem 3, \hat{B} is non-singular and triangular. Hence, the rank of A^0 is m .

Suppose a matrix B of columns of A^0 has such a corresponding graph F . Then by theorem 3, B is non-singular and triangular. Hence, the columns of B are linearly independent, and B is square so has m rows and m columns. Therefore, B is a basis of A^0 .

Suppose that B is a basis of A^0 . From theorems 1 and 2, the proof will be completed if it can be shown that every tree in the forest

F_B has at least one root. Suppose a tree has no root. Then, B has $m-1$ columns or less because a tree has one fewer edge than vertices, and no tree in F can have more than one root. But the rank of A^0 is m so B could not be a basis of A^0 . Hence, the theorem is proven.

Lemma 6 If B is a $m \times m$ non-singular, triangular matrix of 0 , 1 's, and -1 's, and if b is a $m \times 1$ column vector of integers, then the solution to $Bx = b$ has x_j integer for $j=1, \dots, m$.

Proof: The usual iterative method of solving a triangular system of equations is to solve for one variable, substitute its value in its place and move to the right hand side. Then the smaller matrix \bar{B} will be triangular with 0 , 1 , -1 entries, and at each step the variable determined will be an integer. The proof is completed.

Lemma 6 can be used to show that if b , α , and σ are integers, then every basic solution will be integer. Hence, if there is an optimal solution, any basic optimal solution will be all integers. This property can also be proven from the algorithm in section 4 but has been indicated here to complete the discussion of the properties of the matrix B when B is a basis of A^0 .

4. The Simplex Method for Network Flows

In this section, the simplex method for solution of the network flow problem will be presented along with an example. The algebraic details of the simplex algorithm with upper bounds and the use of Phase I and Phase II in solving linear programs are readily available [2] and will not be reviewed here. However, a descriptive outline will be given

as a structure on which the later algorithms will be built. The simplex algorithm begins with a feasible basis and nonbasic variables at upper or lower bounds.

Simplex algorithm

Step 1: Determine values of the dual variables.

Step 2: Price out the variables and select a profitable variable for entry into the basis. If there is no profitable variable, then the present solution is optimal.

Step 3: Determine the changes in value of the basic variables when the new variable is introduced into the basis with the largest change consistent with feasibility. If there is no limit to the change in the new variable, then the objective function is unbounded. Otherwise, go to step 4.

Step 4: If the increase in the new variable is stopped by its reaching its upper or lower bound, then it remains nonbasic at its upper or lower bound, and the algorithm returns to step 2. Otherwise, enter the new variable into the basis and drop from the basis one of the previously basic variables which prevented further change of the entering variable. Define a blocking variable to be a basic variable which becomes infeasible if the entering variable is changed any more. Thus, a blocking variable is dropped from the basis.

This description does not handle the problem of degeneracy, which will be discussed later.

The procedure to be given works directly with the forest F_B in G to carry out these four steps. The concepts introduced here will be used through the remainder of this paper although the

details of carrying out the four steps will differ for different problems.

Let $(v_1, e_1, v_2, \dots, v_{n-1}, e_{n-1}, v_n)$ be a simple path in a network. If an arc $e_i = (v_i, v_{i+1})$, then e_i is called a forward arc in the path, and if $e_i = (v_{i+1}, v_i)$, then e_i is called a reverse arc in the path. In a rooted tree there is a unique simple path from the root to each vertex. An edge will either be a forward arc or a reverse arc in all such paths, and forward arc will be called an up arc with respect to the tree, and a reverse arc will be called a down arc with respect to the tree. Thus, in a rooted forest each arc can be designated as an up arc or a down arc.

Several operations in a rooted forest will be described for later use in the changing of basis in step 4. These operations will not depend on whether the edges are directed or undirected. To reroot a tree means to designate another vertex as its root and drop the old root. To cut off the top of a rooted tree at an edge e , means to delete the edge e from the tree. Then, part of the tree becomes a separate tree which has no root and is called the top of the tree. Either one of its vertices can be designated as the root, or it can be grafted onto a rooted tree by adjoining an edge from it to a rooted tree.

Network Linear Programming Algorithm

Step 1: For a rooted, spanning forest F_B of G , the dual variable

$$\pi_i \text{ at a root } v_i \text{ is given by } \pi_i = \begin{cases} c_i^+ & \text{if } s_i^+ \text{ is basic} \\ -c_i^- & \text{if } s_i^- \text{ is basic} \end{cases} .$$

If π_i is determined, then for an up arc $e_k = (v_i, v_j)$, π_j is given by $\pi_j = -c_k + \pi_i$, and for a down arc $e_k = (v_j, v_i)$, π_j is given by $\pi_j = c_k + \pi_i$. All of the π_i are uniquely determined iteratively because of a rooted tree being connected and having no cycles.

Step 2: To price out and select a new variable for possible entry into the basis, search for an arc e_ℓ or a vertex v_i such that one of the following holds:

- (a) $s_i^+ = 0$ and $\pi_i > c_i^+$;
- (b) $s_i^- = \sigma_i^-$ and $\pi_i > -c_i^-$;
- (c) $s_j^- = 0$ and $\pi_j < -c_j^-$;
- (d) $s_j^+ = \sigma_j^+$ and $\pi_j < c_j^+$;
- (e) $e_\ell = (v_i, v_j)$, $x_\ell = 0$, and $\pi_i - \pi_j > c_\ell$;
- (f) $e_\ell = (v_j, v_i)$, $x_\ell = \alpha_\ell$, and $\pi_j - \pi_i < c_\ell$.

If none of (a)-(f) is found, then the solution is optimal. Otherwise, go to step 3 with such a variable.

Step 3: The feasibility conditions here are $0 \leq x \leq \alpha$, $0 \leq s \leq \sigma$.

In cases (a) and (b) of step 2, let v_1 be the root of the tree containing v_i , and let $P = (v_1, e_1, v_2, \dots, v_{i-1}, e_{i-1}, v_i)$ be the path in F_B from v_1 to v_i .

In cases (c) and (d), let v_r be the root of the tree containing v_j , and let $P = (v_j, e_j, v_{j+1}, \dots, v_{\ell-1}, e_{\ell-1}, v_r)$ be the path in F_B from v_j to v_r .

In cases (e) and (f), suppose that v_i and v_j are in different trees of F_B . Then, let v_1 be the root of the tree containing v_i and v_r be the root of the tree containing v_j . Suppose that the

path from v_1 to v_i is $(v_1, e_1, v_2, \dots, v_{i-1}, e_{i-1}, v_i)$ and the path from v_j to v_r is $(v_j, e_j, v_{j+1}, \dots, v_{\ell-1}, e_{\ell-1}, v_r)$. Let $P = (v_1, e_1, v_2, \dots, v_{i-1}, e_{i-1}, v_i, e_\ell, v_j, e_j, v_{j+1}, \dots, v_{r-1}, e_{r-1}, v_r)$.

For all of the above cases, increase x_k by θ if e_k is a forward arc in P and decrease x_k by θ if e_k is a reverse arc in P . For the first vertex v_1 in P , increase s_1^- by θ or decrease s_1^+ or y_1^+ by θ , whichever is basic or entering there. For the last vertex v_r in P , decrease y_r^- or s_r^- by θ or increase s_1^+ by θ , whichever is basic or entering there. Set θ at the largest value consistent with feasibility and go to step 4. If there is no bound on how large θ can be, then the problem has an unbounded objective function.

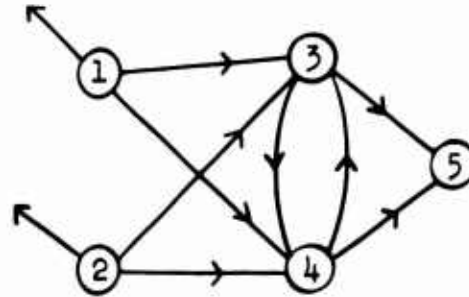
In cases (e) and (f), suppose that v_i and v_j are in the same tree of F_B . Let $P = (v_j, e_j, v_{j+1}, \dots, v_{i-1}, e_{i-1}, v_i)$ be the path in F_B from v_j to v_i . Increase x_k by θ if e_k is a forward arc in P and decrease x_k by θ if e_k is a reverse arc in P . In case (e), increase x_ℓ by θ . In case (f), decrease x_j by θ . As before, set θ at the largest value consistent with feasibility and go to step 4.

Step 4: If the increase in θ was stopped by the new variable reaching its upper or lower bound, then it remains nonbasic, and the algorithm returns to step 2. Otherwise, the new variable enters the basis, and a blocking variable is dropped. We will consider the corresponding arcs or roots as entering F_B and dropping from F_B . There are five cases to consider. The reference is to Example 1, which follows.

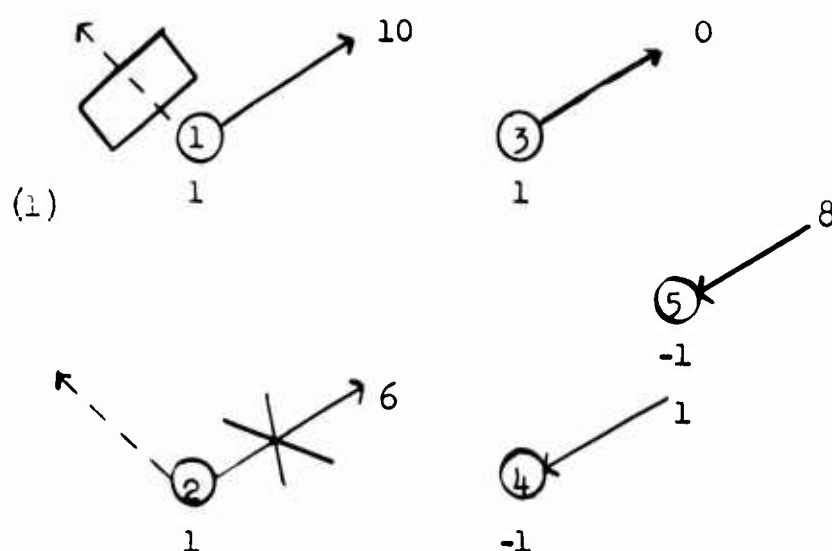
- (a) A root enters, and a root drops. Then the tree is rerooted.
(Example 1, (1)).
- (b) A root enters, and an arc drops. The arc dropping cuts off the top of the tree, and the root enters on the top of the tree.
(Example 1, (4))
- (c) An arc enters from one tree to another, and a root drops. The tree from which the root dropped is grafted onto the other tree by adjoining the entering arc. (Example 1, (2), (3), (6), (8)).
- (d) An arc enters from one tree to another, and an arc drops. The arc dropping cuts off the top of one tree. The top is then grafted onto the other tree by adjoining the entering arc.
(Example 1, (5)).
- (e) An arc enters with both vertices in the same tree. Then, necessarily, an arc drops. The tree is not changed except for addition of one arc and deletion of another arc. (Example 1, (7)).

Example 1 Suppose warehouses 1 and 2 have supplies of 10 and 6 box car loads of thread. Storage capacities are 7 box cars at each warehouse, and storage costs are \$2 per box car per week for warehouse 1 and \$1 per box car per week for warehouse 2. Suppose mills 3, 4, and 5 will need 0, 1, and 8 box car loads of thread in the coming week. The mills have no excess storage space and must meet these demands by shipment from warehouses 1 and 2. The train lines, available space, and costs are shown below in algebraic form. The corresponding graphical form illustrating the routes is shown. The slacks s_1^+ and s_1^- are indicated by arrows out of and into v_1 , respectively. All of the lower bounds are 0.

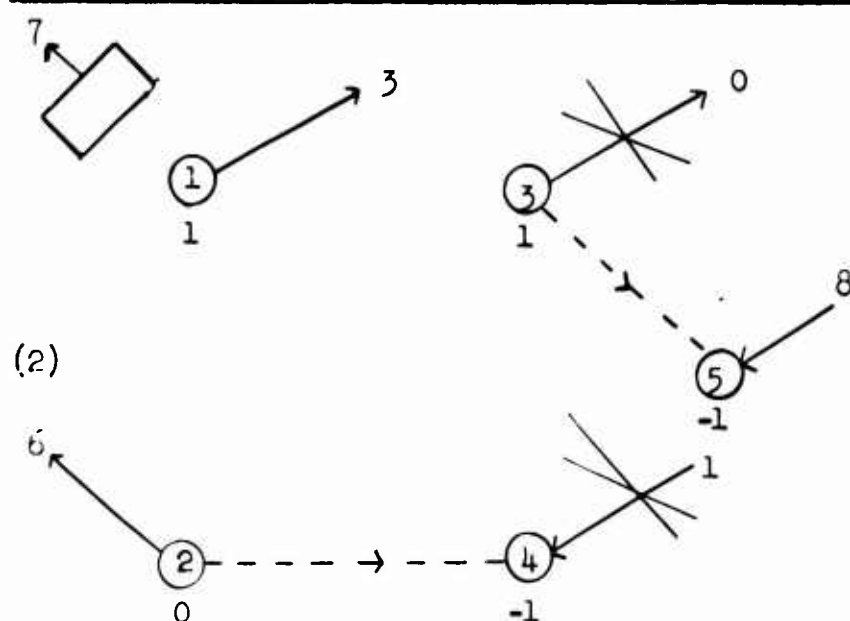
x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	s_1^+	s_2^+	
1		1						1		10
	1		1						1	6
-1	-1			1	-1	1				0
		-1	-1	-1	1		1			-1
						-1	-1			-8
7	5	6	3	2	3	3	6	2	1	costs
5	3	8	8	9	9	5	6	7	7	capacities or upper bounds



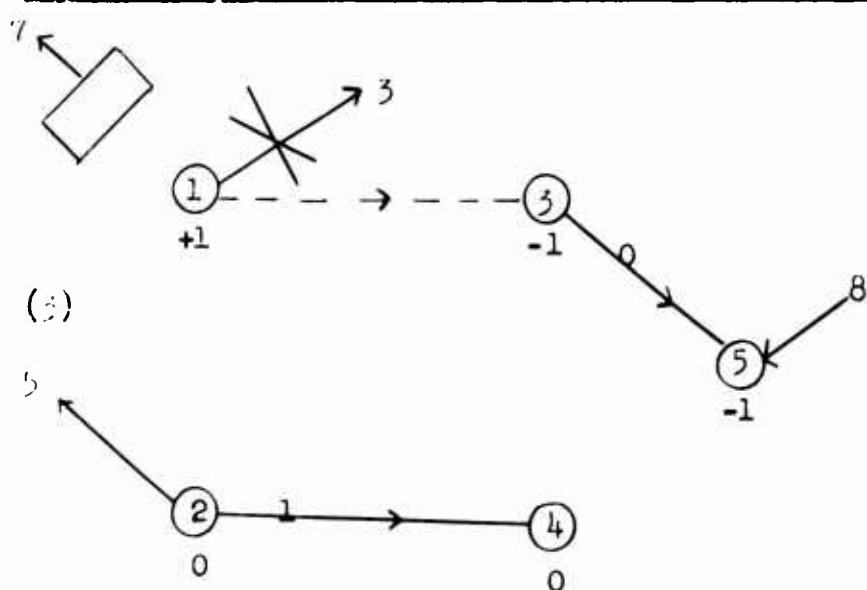
The simplex method starts with phase I, in which all of the cost are 0 except for artificial variables. The entering variables will be indicated by dotted lines, and the variables dropping from the basis will be indicated by X. The symbol $\square \rightarrow$ indicates a variable at upper bound, but the arc is not drawn so that the tree structure of the basis will be clear. The values of the basic variables are indicated by numbers next to the arrows and dual variables by numbers next to the vertices. Each iteration below includes the 5 steps of the simplex algorithm except that the new values of the basic variables are not shown until the next diagram. Two iterations are done in (1) and (2), but should not cause confusion because they are on different components of F_B .



$\pi_1 = 1 > 0$ so s_1^+ can enter basis but reaches its upper bound first. Similarly, s_2^+ can enter and the artificial at v_2 drops.

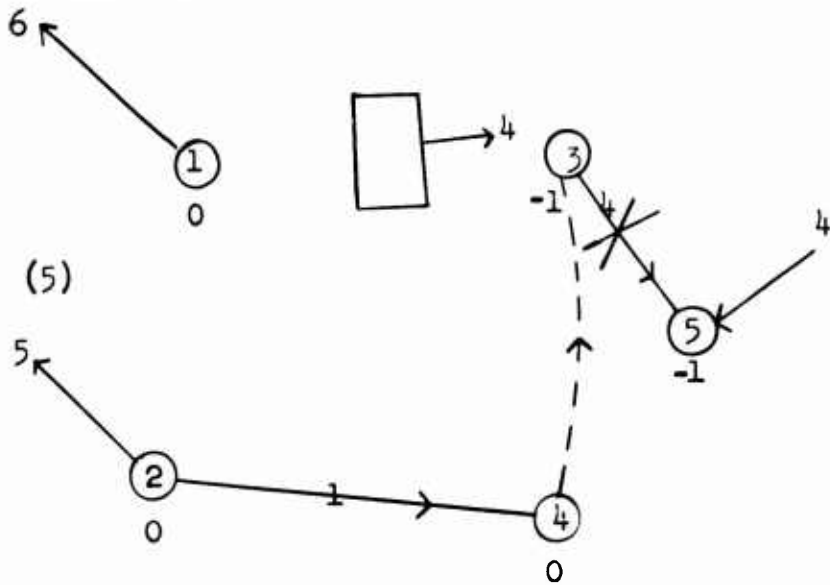
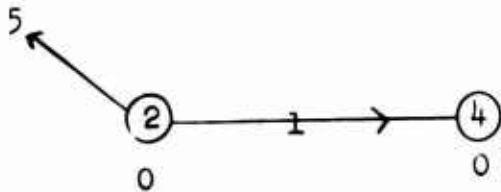


The new basic and dual values are shown. s_1^+ is upper bounded at 7. $\pi_2 - \pi_4 = +1 > 0$ and $x_4 = 0$, and $\pi_3 - \pi_5 = -2 > 0$ and $x_7 = 0$ so x_4 and x_7 enter. Two artificials drop.

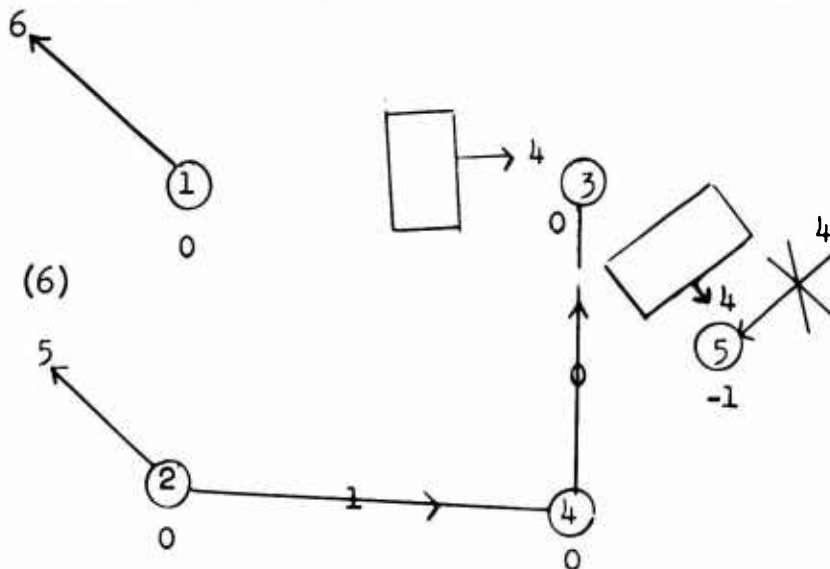


$\pi_1 - \pi_3 = +2 > 0$ and $x_1 = 0$ so x_1 enters, and the artificial at v_1 drops.

(4)

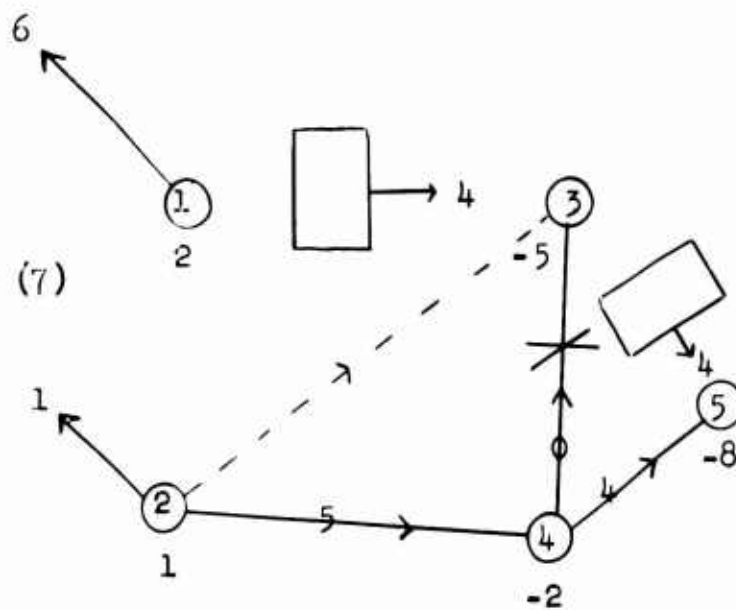


(5)



(6)

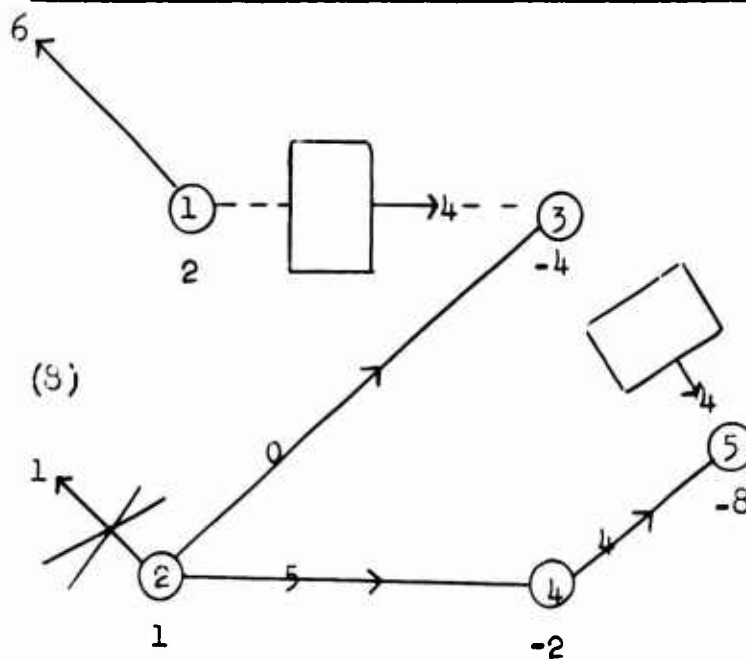
END OF PHASE I



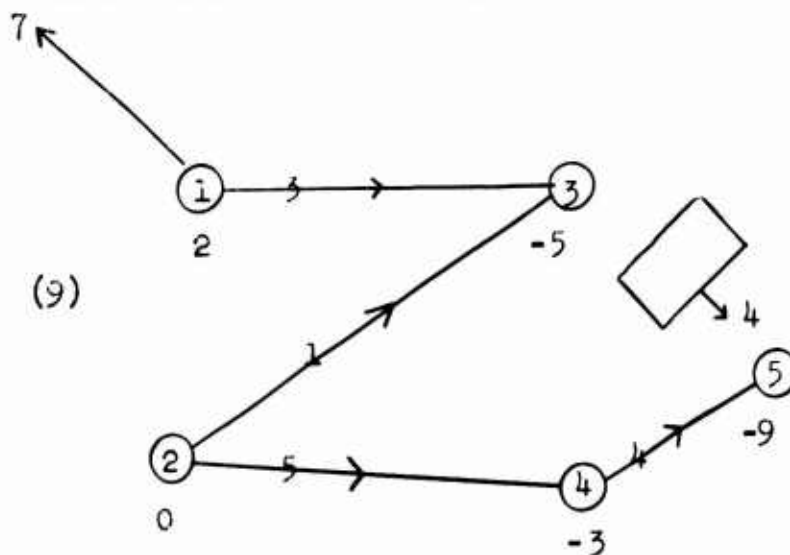
The dual variables are now calculated using costs

$$c_k \cdot \pi_2 - \pi_3 = +6 > 5$$

and $x_2 = 0$ so x_2 enters, and x_6 drops.



$\pi_1 - \pi_3 = 6 < 7$ and $x_1 = \beta_1$ so x_1 enters, and s_2^+ drops.



Pricing out now reveals that an optimum solution has been reached.

For phase I, let us denote the dual variables by ρ_i instead of π_i and let $\bar{d}_i = -(\rho_i - \rho_j)$ when $e_k = (v_i, v_j)$, $\bar{d}_i^+ = -\rho_i$, and $\bar{d}_i^- = \rho_i$. Theorem 4 says the matrix B is a basis of A^0 if, and only if, F_B is a rooted, spanning forest of G , provided that G has a vertex corresponding to a column of U for each connected component of G . Phase I starts with artificials, which are the same as slacks singled out to be minimized, at all of the vertices of G so G does have a vertex with slack in each connected component. The simplex method goes from basis to basis, and the rank of every basis is the same provided we go from phase I to phase II by fixing the values of and eliminating from further consideration those variables at zero with $\bar{d} > 0$ and those variables at upper bounds with $\bar{d} < 0$. Theorems 1 and 2 assure that F_B will always be a spanning forest of G with at most one root on each tree for B a basis of A^0 . If some tree did not have a root, then the basis would have lower rank than the original basis contradicting the above statement which always holds for the simplex method. Hence, the assumption that every connected component of G has a vertex with a slack is justified because it always holds in the computational procedure.

A labeling procedure can easily be devised to assist in carrying out the computation as described. However, an efficient labeling procedure for this algorithm must be able to go up the tree as well as down the tree. Some promising work along these lines has been done by Scions [10] and others.

The algorithm given does not resolve the degeneracy problem.

That is, in step 4 if several variables are blocking variables, how do we decide which one to drop? In example 1, this question arises in iteration (4). In practice, the algorithm is almost always finite without any special procedure for resolving degeneracy. The next section treats degeneracy in a better way than can usually be done.

5 Phase I

Phase I of the network flow problem is called the max-flow problem. Roughly, this name originates by considering the artificials as being a deficit flow, and the problem being to maximize flow or minimize this deficit.

To begin, set the variables x , s at zero. Let $y_i^+ = b_i$ if $b_i > 0$ and $y_i^- = -b_i$ if $b_i < 0$. The costs d are 0 for all variables except artificials and $d_i^+ = d_i^- = 1$ for artificials y_i^+ and y_i^- . The dual variables are denoted by p_i . The artificials are actually slacks which are distinguished by having $d = 1$.

Steps 1, 2, and 3 of the simplex algorithm offer some obvious simplifications for this max-flow problem. Step 4 can be modified to resolve the degeneracy problem in a much better way than is generally available for a linear program and even better than is available for the network flow problem. One of the modifications is the use of slacks $s_i^+ = 0$ which have d_i^+ equal to zero, but were not originally present in the problem so have σ_i^+ equal to zero. These slacks are dummy slacks added to give a root to a tree, but they must remain zero because no slack is permitted there. Once they become nonbasic, they are dropped. To begin, if $b_i = 0$, let a slack be basic at v_i if there is a slack there. If not, adjoin a slack

s_1^+ with $\sigma_1^+ = 0$. The rooted, spanning forest F_B consists now of trees consisting of one vertex which is the root of the tree.

Max-Flow Algorithm

Step 1: All of the ρ_i are +1, 0, or -1 depending on whether the tree containing v_i has a root v_j with y_j^+ , a slack, or y_j^- . Hence, determining the dual variables ρ_i is simply a matter of determining the root of the tree containing the vertex v_i and is no longer an arithmetic operation.

Step 2: Let $V_1 = \{v_i \mid \rho_i = +1\}$, $V_2 = \{v_i \mid \rho_i = 0\}$, and $V_3 = \{v_i \mid \rho_i = -1\}$. Then, pricing out and selecting a new variable is reduced to searching for one of the following:

- (a) $v_i \in V_1$, and $s_i^+ = 0$ or $s_i^- = \sigma_i^-$;
- (b) $v_j \in V_3$, and $s_j^+ = \sigma_j^+$ or $s_j^- = 0$;
- (c) $e_\ell = (v_i, v_j)$, $x_\ell = 0$, and $v_i \in V_1, v_j \in V_3$, or
 $v_i \in V_1, v_j \in V_2$, or
 $v_i \in V_2, v_j \in V_3$;
- (d) $e_\ell = (v_j, v_i)$, $x_\ell = \alpha_\ell$, and $v_i \in V_1, v_j \in V_3$, or
 $v_i \in V_1, v_j \in V_2$, or
 $v_i \in V_2, v_j \in V_3$.

Hence, pricing out as well as determining the dual variables, is not an arithmetic process, but is only a search. Note that cases (a) and (b) of step 2 in the network linear programming algorithm have now been collapsed into case (a) and cases (c) and (d) there into case (b). If none of (a)-(d) is found, then the solution is optimal. Otherwise, go to step 3 with such a variable.

Step 3: Step 3 is the same as for the network linear programming algorithm except that an entering arc will never have both vertices in the same tree because an entering arc always has vertices in different V_1 . Hence, the flow will change by amount θ along a path P from y_1^+ to y_j^- , or from y_1^+ to a slack, or from a slack to y_1^- . Since $y_1 \geq 0$ is now part of the feasibility condition, there is always a bound on how large θ can be set without violating feasibility. The path P is called a flow segmenting path because the change in flow results in a decrease in the artificials at the end of P .

Step 4 (1): Step 4(1) is the same as step 4 of the network linear programming algorithm except that case (e) there never occurs here, and here we specify in certain cases which of the blocking variables to drop.

If an artificial y_1^+ or y_1^- reaches zero in step 3, then change it to a slack $s_1^+ = 0$ with $\sigma_1^+ = 0$.

If P has two slacks at its ends, then any blocking variable can be dropped. If P has one slack and one artificial, then drop the blocking variable nearest in P to the artificial. If P has an artificial at each end and only one variable is a blocking variable, then drop it. Otherwise, there are at least two blocking variables corresponding to arcs in P . Drop the blocking arc nearest one end of P and the blocking arc nearest the other end of P . On a vertex v_1 of P between these two arcs, put a slack $s_1^+ = 0$ with $\sigma_1^+ = 0$. Now, there are three rooted trees where there were two rooted trees before.

Step 4(ii): If in step 4(i), a tree with an artificial y_1^+ at the root v_1 had another tree or top of a tree grafted onto it, then begin at the edge adjoined in the grafting and trace the tree away from the root along every branch until either an end of the tree is reached, or an up-arc $e = (v_1, v_2)$ at upper bound is reached, or a down-arc $e = (v_2, v_1)$ at lower bound is reached. If either of the two types of arcs e is found, cut off a top of the tree by deleting e from F_B and make v_2 a root of the top with slack at zero and upper bound of zero.

Repeat for artificials y_1^- except look for up-arcs at lower bound and down-arcs at upper bounds.

Return to step 1.

Proof of finiteness of the algorithm: The observation in step 3 that the flow augmenting path P is from v_1 with y_1^+ to v_j with y_1^- , or from y_1^+ to a slack, or from a slack to y_1^- is important because it shows that the flow change along P in step 3 is always away from y_1^+ and toward y_1^- . But step 4(ii) assures that at any iteration there can always be a positive flow change away from the root of a tree with root y_1^+ and toward the root of a tree with root y_1^- . If any iteration does not result in a change in flow, then the blocking variable cannot be the entering arc or an arc in the tree with an artificial so the tree with an artificial grows by at least the entering arc. Hence, the number of elements in V_2 decreases at each iteration which does not result in a change in flow. But, there are only m vertices in all so there can be at most m iterations

in a row resulting in no change in flow.

If α , σ , and b are integer, then each change in flow results in an integer decrease in the sum of the artificial variables. If the original sum of the artificials is M , then the algorithm cannot take more than $m M$ iterations because the artificials are non-negative.

Even with fractional and irrational α , σ , and b , finite convergence of the algorithm can be proven. The variables in the forest F_B are determined once all the other variables are determined. But the other variables can take on only two values, their upper and lower bounds. Hence, a given forest can only take on a finite number of values and, hence, can only recur a finite number of times. There are a finite number of rooted, spanning forests so the algorithm is finite.

The reason the Ford-Fulkerson labeling procedure for the max-flow problem may not be finite (page 21[]) when α , σ , and b are irrational is that the variables strictly between their upper and lower bounds may not form a forest; that is, they may form cycles. This difficulty can be overcome within the framework of that procedure by forming the set E_1 of arcs between their upper and lower bounds. After each breakthrough, the labeling procedure is done first in E_1 and then using arcs not in E_1 but checking all of the arcs in E_1 each time a new vertex is labeled and unscanned. Then, the arcs between their bounds never form a cycle. For a network with one source, one sink, and no slacks, the labeling procedure with this modification is a way of accomplishing the above algorithm except that

the trees in V_2 are not kept track of. The labeling procedure destroys all the labels after a change in flow. Whether a more efficient labeling procedure can be devised to modify labels with change in flow is yet to be seen.

The Ford-Fulkerson procedure converts a general network flow problem to a problem with one source, one sink, and no slacks. This conversion was not done here because the purpose is to develop the network flow theory within the framework of linear programming in order to show the connection. Adjoining the additional source, sink, and slack vertex makes sense graphically, but not algebraically, and the graphical procedure without adjoining additional vertices is closer to the linear programming procedure. Note also that positive lower bounds are handled easily because no assumption is made on b .

In summary, the advantages of the max-flow algorithm over the network linear programming algorithm are simplicity in determining dual variables and pricing out, and the fact that at most m iterations in sequence can occur with no change in flow. This fact is a result of having no arcs enter with both ends in the same tree because when that happens the flow change is toward the root in some arcs and away from the root in others. The flow change is always away from a root v_1 with y_1^+ and toward a root v_1 with y_1^- . The trees attached to roots with artificials are constructed so that such a flow change is possible.

6. The Primal-Dual Method for Network Flows

The primal-dual method was devised [6] for the network flow problem, and then it was generalized to the general linear program

[5]. Its advantage over the simplex method depends on some simplification for the restricted phase I problem. For network flows, its advantages over the network linear programming algorithm are the same as the advantages of the max-flow algorithm just given.

For a network programming problem, define

$$(1) \text{ dual feasibility conditions: } \pi_i - \pi_j < \begin{cases} \leq c_k, & \text{if } e_k = (v_i, v_j) \text{ and } x_k = 0 \\ \geq c_k, & \text{if } e_k = (v_i, v_j) \text{ and } x_k = \alpha_k \\ \alpha_k \end{cases}$$

$$\pi_i < \begin{cases} \leq c_i^+, & \text{if } s_i^+ = 0 \\ \geq c_i^+, & \text{if } s_i^+ = \sigma_i^+ \\ \geq -c_i^-, & \text{if } s_i^- = 0 \\ \leq -c_i^-, & \text{if } s_i^- = \sigma_i^- \end{cases}$$

The primal-dual method starts with a π , x , and s satisfying the dual feasibility conditions and $0 \leq x \leq \alpha$, $0 \leq s \leq \sigma$, but not $Ax + Us = b$. If $c \geq 0$, then $\pi_i = 0$ all i and $x_k = s_i^+ = s_j^- = 0$ is such a π , x , and s . If c_k , c_i^+ or c_i^- is negative, set the corresponding variable at upper bound. If there is no upper bound, then set an upper bound M and let the variable be equal to M . If at the conclusion the variable is still equal to M , then raise the upper bound M , introduce new artificials, and solve again.

Starting with a π , x , and s satisfying (1), and $0 \leq x \leq \alpha$, $0 \leq s \leq \sigma$, the primal-dual method uses the max-flow algorithm restricted to arcs $e_k = (v_i, v_j)$ and slacks s_h^+ and s_ℓ^- such that $\pi_i - \pi_j = c_k$, $\pi_h = c_h^+$, and $\pi_\ell = -c_\ell^-$. At the conclusion of the

max-flow subroutine, the π 's are changed to $\pi_1 + \epsilon$, $v_1 \in V_1$,
 and $\pi_1 - \epsilon$, $v_1 \in V_3$, where ϵ is chosen as large as possible without
 violating the dual-feasibility conditions (1). Note that $\epsilon > 0$.

The max-flow subroutine is the primal step, and the dual change
 is the dual step. The two alternate until either no artificials
 remain in the primal step or until there is no bound on how large ϵ
 can be chosen in the dual step. If there are no remaining artificials,
 then the solution π, x, s is optimal. If there is no bound on
 ϵ , then there is no solution to $Ax + Us = b$, $0 \leq x \leq \alpha$, $0 \leq s \leq \sigma$.

7. Flows with Gains

A network with gains is defined to be a network G together with a function w on E , the edges of G , to the reals. Then, $w(e)$ is the gain associated with the edge e . Assume $w(e) \neq 0$.

$$\text{Define } a_{ik} = \begin{cases} 1 & \text{if arc } e_k = (v_i, v) \text{ for some } v \in V \\ w(e_k) & \text{if arc } e_k = (v, v_i) \text{ for some } v \in V \\ 0 & \text{otherwise} \end{cases}$$

The linear program in a network with gains or flows with gains [8] problem is:

$$\begin{aligned} (2) \quad & Ax + Us = b, \quad 0 \leq x \leq \alpha, \quad 0 \leq s \leq \sigma, \\ & cx + cs = z(\min). \end{aligned}$$

If $w(e_k) = -1$ for all arcs e_k , the problem is the same as in sections 1-6. If $w(e_k) > 0$ and the graph is bipartite, Dantzig [2] has called the problem the weighted distribution problem and given the structure of the basis. Here the basis is slightly more complex than in sections 1 through 6. First, a preliminary lemma is needed.

Lemma 7 A connected graph G with k vertices and k edges has one cycle, and if the edges of the cycle are removed, then the remaining graph is a forest with each tree having exactly one vertex in the graph.

Proof: The graph G must have at least one cycle since, otherwise, G would be a tree, and a tree with k vertices has $k-1$ edges by lemma 4.

Suppose G has two cycles. Then, there is an edge of G in one cycle but not in the other. The removal of that edge from G does not destroy connectedness and leaves one cycle in G . Then, G has k vertices and $k-1$ edges so is a tree by lemma 4; but a tree has no cycles so a contradiction is reached.

If all of the edges of the cycle are removed, then the remainder of the graph is a forest because it has no cycles. Each tree of the forest must include a vertex of the cycle because the original graph G was connected. Suppose there were two vertices of the cycle in some tree of the remaining forest. Then, there is a path in the tree between the two vertices and a path in the cycle between the two vertices. Together, they form a cycle. Hence, the original graph G would have had two cycles contradicting the previous part of the lemma. Thus, the lemma is proven.

Let $A^0 = [A, U]$.

Theorem 5 Let A^0 have rank m . The connected components of the graph H_B corresponding to a basis B of A^0 is either a rooted tree, or a graph with the same number of vertices and edges and having no slack.

Proof: A proof similar to the proof of theorem 2 shows that H_B can have at most one slack corresponding to a connected component. The same idea as before of sending one unit of flow from one slack to the other along a simple path is still applicable, but the algebraic details are more complicated and are omitted here.

If a connected component of H_B on k vertices has $k+1$ or more corresponding columns of B , then those $k+1$ or more columns

have only k rows with non-zero entries, contradicting independence of the columns of B .

Therefore, if a connected component of H_B with k vertices has one slack, then it cannot have more than $k - 1$ edges. It must have $k - 1$ edges to be connected. Hence, the component is a rooted tree. If a connected component of H_B with k vertices has no slack, then it cannot have more than k edges. But A^0 and, hence, B have rank m , so each connected component must have as many corresponding columns of B as vertices. Hence, a connected component with k vertices and no slack must have k edges, and the proof is completed.

In sections 3-6 the basis was always triangular, and the entries were $+1$ and -1 . Here, even when the basis is triangular, the entries are not $+1$ and -1 but are 1 and $w(e_k)$. The changes in steps 1 - 3 of the network programming algorithm given in section 4, due to the gains $w(e_k)$ will be discussed first under the assumption that the graph H_B has no cycles; that is, H_B is a rooted forest.

Step 1: At the root, π_1 is still equal to c_1^+ or $-c_1^-$. If π_1 is known and $e_k = (v_1, v_j)$ is an up-arc, then $\pi_j = \frac{1}{w(e_k)} (c_k - \pi_1)$. If π_1 is known and $e_k = (v_j, v_1)$ is a down-arc, then $\pi_j = c_k - w(e_k)\pi_1$.

Step 2: Pricing out is the same for the slacks, but for the arcs it changes to:

$$(e) \quad e_\ell = (v_1, v_j), \quad x_\ell = 0, \quad \text{and} \quad \pi_1 - w(e_\ell)\pi_j > c_\ell;$$

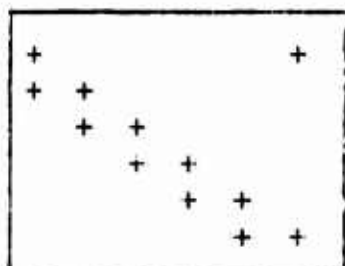
$$(f) \quad e_\ell = (v_j, v_1), \quad x_\ell = \alpha_\ell, \quad \text{and} \quad \pi_j - w(e_\ell)\pi_1 < c_\ell.$$

Step 3: This step is similar to step 3 of the network programming algorithm except for two differences.

Previously, the flow change was $+\theta$ or $-\theta$ in all of the arcs of P . Now, suppose $P = (v_1, e_1, v_2, \dots, e_{k-1}, v_k, e_k, v_{k+1}, e_{k+1}, \dots, v_{r-1}, e_{r-1}, v_r)$, and x_k changes by θ . If e_k is a forward arc in P , then x_{k-1} changes by $-\frac{1}{w(e_{k-1})}\theta$ if e_{k-1} is a forward arc in P and by $+\theta$ if e_{k-1} is a reverse arc in P ; and x_{k+1} changes by $+\frac{w(e_k)}{w(e_{k+1})}\theta$ if e_{k+1} is a forward arc in P and by $-\frac{w(e_k)}{w(e_{k+1})}\theta$ if e_{k+1} is a reverse arc in P . If e_k is a reverse arc in P , then the change in x_{k-1} is the same as the change in x_{k+1} above, and the change in x_{k+1} is the same as the change in x_{k-1} above. The change in slack at v_1 is either $\pm\theta$ or $\pm w(e_1)\theta$ if x_1 changes by θ and similarly for v_r .

The second difference occurs when an entering arc has both vertices in the same tree. Then a flow change can take place not only around a cycle, but down the path from the cycle to the root. If the variable dropping is on the path instead of the cycle, then the cycle becomes part of the basis.

A cycle corresponds to a matrix of the form:



where $+$ indicates a non-zero entry.

For any right-hand side, a system of equations with such a coefficient matrix can be solved by considering the first variable to be a

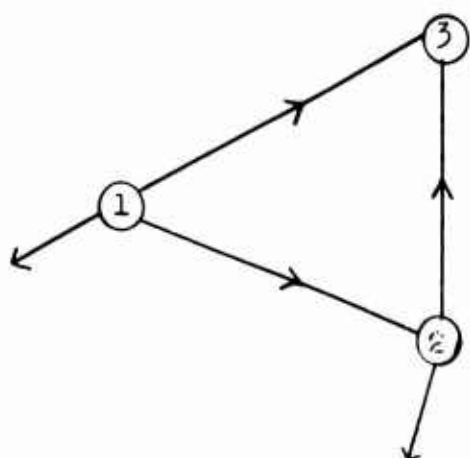
parameter, moving it to the right hand side, and solving the remaining system, which is triangular.

If the cycles are thought of as being a single vertex, then the connected component becomes a rooted tree with the cycle being the root. In step 1, computing the dual variables, the π_1 around the cycle can be solved first, and in step 3, computing the change in flow, the flow change around the cycle can be computed last as a separate subroutine as indicated above. Step 4, the change in basis, is similar to the network programming algorithm except that whenever a slack enters or drops there, a cycle could enter or drop here in the sense that an arc enters forming a cycle or an arc drops destroying a cycle. The cycles can form when an arc enters with both ends in the same tree as mentioned in step 3 above.

Phase I for this problem does simplify somewhat. The tree without artificials, including the trees with cycles, all have dual variables $\rho_1 = 0$ at every vertex v_1 . The only computation of dual variables is on trees with an artificial, but, there, actual computation must be done. Pricing out does not simplify, but every entering arc will have at least one end in a tree with an artificial variable at the root. Hence, any change of flow involving a cycle will be along a path with the cycle at one end and an artificial variable at the other end.

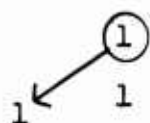
However, the handling of degeneracy in the max-flow algorithm in sections 1-6 can not be done for this problem. For the max-flow algorithm, the flow change was always away from an artificial y_1^+ and toward an artificial y_1^- . That this important property no longer holds is illustrated by the example below.

Example 1 The network is illustrated below, and the algebraic statement of the problem gives the edge weights.



	x_1	x_2	x_3	y_1^+	y_2^+	
1	1	1		1		= 1
2	$-\frac{1}{2}$		1		1	= 1
3		-2	-10			= 0
	0	0	0	1	1	

(1)

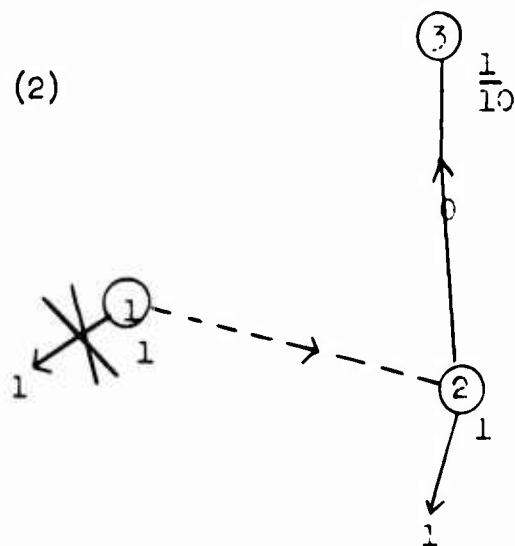


No upper bounds are placed.

s_3^+ is added but restricted to remain 0.

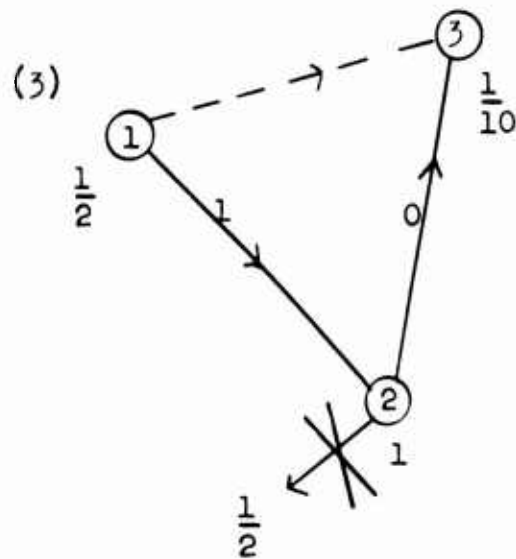
$\pi_2 - 10\pi_3 = 1 > 0$ so x_3 becomes basic, and y_3^+ drops.

(2)

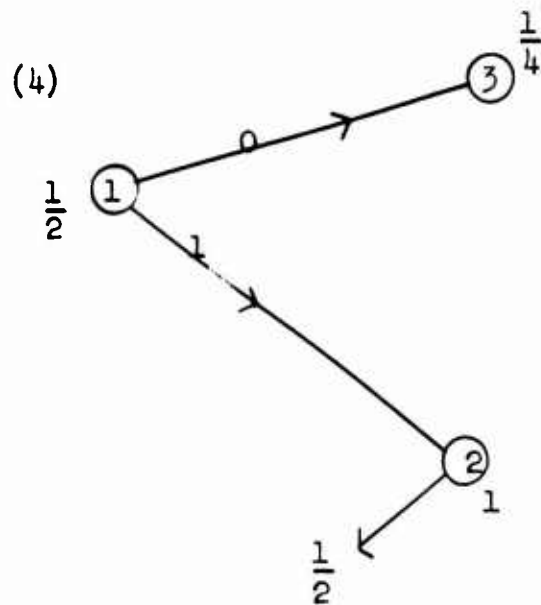


$\pi_1 - \frac{1}{2}\pi_2 = 1 - \frac{1}{2} > 0$ so x_1

enters, and y_1^+ drops



$\pi_1 - 2\pi_3 = \frac{1}{2} - \frac{1}{5} > 0$ so x_3
enters, and x_2 drops



Optimal

Iteration (3) illustrates a case when the change in flow in arc (v_2, v_3) is toward the basic artificial y_2^+ .

The two advantages of the max-flow algorithm do not carry over to the phase I problem here, except that the dual variables on a tree without artificial variables are zero, and the cycles only enter into a basis change at one end of a path. The primal-dual method can be used here, but lacks any special advantage since the phase I problem

does not significantly simplify. However, there is a special case of the flows with gains problem in which the Phase I problem simplifies as much as it did in the previous chapter. The next section discusses that problem.

8. Linear Programming in an Undirected Graph

Suppose in the flows with gains problem, all of the gains $w(e_k) = +1$. Then the matrix A is a vertex-edge incidence matrix of an undirected graph G . The columns of A correspond to edges and have two $+1$ entries indicating the two incident vertices. The edges are undirected and will be written $e_k = [v_i, v_j]$ to distinguish them from directed edges or arcs. The order of v_i, v_j has no importance; there is at most one edge between v_i and v_j , and $[v_i, v_j]$ represents the same edge as $[v_j, v_i]$.

If any $b_i < 0$, then there must be a s_i^- because, otherwise, there would be no feasible solution to $Ax + Us = b$. Hence, the i^{th} row of $Ax + Us = b$ is $\sum_{k=1}^n a_{ik} x_k - s_i^- = b_i$, and $c_i^- s_i^- = -c_i^- b_i + \sum_{k=1}^n (c_i^- a_{ik}) x_k$.

Therefore c_i^- can be replaced by zero and c_k by $c_k + c_i^- a_{ik}$. If $\sigma_i^- = \infty$, then $\sum_{k=1}^n a_{ik} x_k \geq b_i$ holds for all feasible x and, hence, the i^{th} row of $Ax + Us = b$ can just be dropped from the system. If $\sigma_i^- < \infty$, then adding σ_i^- to both sides of $\sum_{k=1}^n a_{ik} x_k - s_i^- = b_i$ gives $\sum_{k=1}^n a_{ik} x_k + (\sigma_i^- - s_i^-) = b_i + \sigma_i^-$. Now, $\sigma_i^- - s_i^- \geq 0$ so if $b_i + \sigma_i^- \leq 0$, then there is no feasible solution. If $b_i + \sigma_i^- \geq 0$, replace b_i by $b_i + \sigma_i^-$, replace s_i^- by $s_i^+ = \sigma_i^- - s_i^- \geq 0$, and set $\sigma_i^+ = b_i + \sigma_i^-$. Therefore, only the case $b \geq 0$ need be considered here. All of the artificial variables will be y_i^+ so we drop the $+$ and just write y_i .

Define the length of a cycle to be the number of edges in it.

Lemma 8 If B is a basis of A^0 , then the cycles in H_B are of odd length.

Proof: Let $(v_1, e_1, v_2, \dots, v_{2k}, e_{2k}, v_1)$ be a cycle of even length.

Then $a_{ij} = \begin{cases} 1 & , \text{ if } e_j = [v_i, v], \text{ any } v \\ 0 & , \text{ otherwise} \end{cases} = \begin{cases} 1, & \text{ if } i=j, i=j+1, \text{ or } i=1 \\ & \text{ and } j=2k. \\ 0, & \text{ otherwise.} \end{cases}$

Let $x_j = 1$ for j even and $x_j = -1$ for j odd. Then $\sum_{j=1}^{2k} a_{ij}x_j = 0$ for all i because for $i \geq 2$, $\sum_{j=1}^{2k} a_{ij}x_j = a_{i,i-1}x_{i-1} + a_{ii}x_i = x_{i-1} + x_i = 0$, and for $i = 1$, $\sum_{j=1}^{2k} a_{1j}x_j = a_{11}x_1 + a_{1,2k}x_{2k} = -1 + 1 = 0$. Hence,

a basis could not include an even cycle because the columns corresponding to an even cycle are linearly dependent.

If G has no odd cycles, then every basis has no cycle so H_B is a rooted forest. A graph G with no odd cycles is called bipartite and is easily proven to have the property that the vertices can be divided into two disjoint sets V_1, V_2 such that every edge is incident to one vertex of V_1 and one of V_2 . The rows of A^0 corresponding to vertices of V_2 can be multiplied by -1 converting A^0 into the network flow type matrix. This section is, therefore, mainly concerned with graphs G having odd cycles.

Each odd cycle in the basis will be considered to be a single vertex so that the resulting graph F_B corresponding to B is a rooted forest.

For a rooted tree, define the distance from the root to a vertex to be the number of edges in the path from the root to the vertex.

For example, the distance from the root to itself is zero. Define the distance from the root to an edge to be the number of edges in the path from the root to the vertex of the edge nearest the root. For example, an edge incident to the root is distance zero from the root. The edges and vertices are called even or odd according as the distance from the root is even or odd.

An algorithm similar to the max-flow algorithm will be given for Phase I of this problem. To begin, set the variables x , s at zero. Just as in the max-flow algorithm, let $y_i = b_i$ if $b_i > 0$, and if $b_i = 0$ let a slack be basic at v_i if there is a slack there. If not, set $s_i^+ = 0$ and $\sigma_i^+ = 0$.

For a tree with root v_1 and y_1 basic, the even edges are called increasing edges, and the odd edges are decreasing edges. Increasing and decreasing edges are only defined in trees with an artificial at the root.

Alternating Path Algorithm

Step 1: The dual variables ρ_i are zero on trees without an artificial.

On a tree with root v_1 and artificial y_1 , $\rho_j = +1$ for v_j an even vertex and $\rho_j = -1$ for v_j an odd vertex.

Step 2: Let $V_1 = \{v_i \mid \rho_i = +1\}$, $V_2 = \{v_i \mid \rho_i = 0\}$, and $V_3 = \{v_i \mid \rho_i = -1\}$.

Pricing out and selecting a new variable amounts to searching for one of the

- following:
- (a) $v_i \in V_1$ and $s_i^+ = 0$ or $s_i^- = \sigma_i^-$;
 - (b) $v_j \in V_3$ and $s_j^+ = \sigma_j^+$ or $s_j^- = 0$;
 - (c) $e_\ell = [v_i, v_j]$, $x_\ell = 0$, and $v_i \in V_1, v_j \in V_1$ or $v_i \in V_1, v_j \in V_2$;
 - (d) $e_\ell = [v_i, v_j]$, $x_\ell = \alpha_k$, and $v_i \in V_3, v_j \in V_3$ or $v_i \in V_3, v_j \in V_2$.

The v_1, v_j are not in any order so which of v_1, v_j is in V_2 and which is in V_1 or V_3 does not matter.

Step 3: The feasibility conditions are $0 \leq x \leq \alpha$, $0 \leq s \leq \sigma$, $0 \leq y$.

In case (a) of step 2, let v_1 be the root of the tree containing v_i . Since $v_i \in V_1$, v_1 has an artificial variable because all of the trees with artificial variables have all of their vertices in V_2 . Let $P = (v_1, e_1, v_2, \dots, v_{i-1}, e_{i-1}, v_i)$ be the path in F_B from v_1 to v_i .

In case (b) of step 2, the tree containing v_j has a root v_r with an artificial just as in case (a). Let $P = (v_j, e_j, v_{j+1}, \dots, v_{r-1}, e_{r-1}, v_r)$ be the path in F_B from v_j to v_r .

In cases (c) and (d), suppose v_1 and v_j are in the same tree of F_B . Then v_1 and v_j are both in V_1 or both in V_3 so the root v_1 has an artificial. Let the path in F_B from v_1 to v_j be $(v_1, e_1, v_{i+1}, \dots, v_{j-1}, e_{j-1}, v_j)$. Form the cycle $(v_1, e_1, v_{i+1}, \dots, v_{j-1}, e_{j-1}, v_j, e_\ell)$ and let $P = (v_1, e_1, v_2, \dots, v_{r-1}, e_{r-1}, v_r)$ be the path F_B from v_1 to the cycle, where v_r is a vertex of the cycle.

In cases (c) and (d), suppose v_1 and v_j are in different trees of F_B . Let v_1 be the root for the tree containing v_i and v_r be the root of the tree containing v_j . Suppose the path in F_B from v_1 to v_i is $(v_1, e_1, v_2, \dots, v_{i-1}, e_{i-1}, v_i)$ and from v_j to v_r is $(v_j, e_j, v_{j+1}, \dots, v_{r-1}, e_{r-1}, v_r)$. Let $P = (v_1, e_1, v_2, \dots, v_{i-1}, e_{i-1}, v_i, e_\ell, v_j, e_j, v_{j+1}, \dots, v_{r-1}, e_{r-1}, v_r)$. One end of P could be a cycle, but at least one end of P must be an artificial. In case (c), designate e_ℓ to be an increasing edge,

and in case (d) designate e_ℓ to be a decreasing edge. If only one end of P has an artificial, then starting at e_ℓ and going toward the end of P without artificial, designate the edges alternately as increasing and decreasing edges.

Suppose an end of P is a cycle. Then that end edge in P is incident to a vertex v_1 of that cycle. Denote the cycle $(v_1, e_1, v_2, \dots, v_{2k+1}, e_{2k+1}, v_1)$. Designate e_1 as an increasing edge if the edge in P incident to the cycle is decreasing, and e_1 is a decreasing edge if the edge in P is increasing. Going around the cycle, alternately designate the edges as increasing or decreasing. The edge e_{2k+1} is designated the same as e_1 because the cycle is of odd length.

Increase x_k by $\frac{\theta}{2}$ if e_k is an increasing edge in a cycle and decrease x_k by $\frac{\theta}{2}$ if e_k is a decreasing edge in a cycle. Increase x_k by θ if e_k is an increasing edge of P and decrease x_k by θ if e_k is a decreasing edge of P . Decrease artificials at the end of P by θ . If a slack s_i^+ is basic at the end of P , then increase or decrease s_i^+ by θ according as the edge in P incident to the end of P is decreasing or increasing. If a slack s_i^- is basic at the end of P , then increase or decrease s_i^- by θ according as the edge in P incident to the end of P is increasing or decreasing. Decrease y_i by θ if v_i is an end of P . Set θ as large as possible, consistent with feasibility.

Step 4(1): The types of basis change have been discussed in section 1.

If an artificial reaches zero in step 3, then replace it by a slack with zero upper bound.

If P now has artificials at neither end, any blocking variable

can be dropped. If P has exactly one artificial, then drop the blocking variable nearest in P to the artificial. If no edge of P is blocking, and the other end of P has a vertex-cluster, then drop any blocking edge of the vertex-cluster. If P has an artificial at each end and only one blocking variable, then drop it. Otherwise, there are at least two blocking arcs in P . Drop the blocking arc nearest one end of P and drop the blocking arc nearest the other end of P . On a vertex v_1 of P between these two arcs, put a slack $s_1^+ = 0$ and $\sigma_1^+ = 0$. Now there are three rooted trees where there were only two rooted trees.

Step 4(ii): If in step 4(i) a tree with artificial at the root had another tree or a top of a tree grafted onto it, then begin at the edge adjoined in the grafting and trace the tree away from the root along every branch until an increasing edge at upper bound or a decreasing edge at lower bound is reached. Cut off a top of the tree by deleting the edge from F_B and put a root on the top with slack at zero and upper bound of zero.

Return to step 1.

The algorithm is completed. Now its correctness and finiteness will be proven.

Lemma 9 The increasing and decreasing edges alternate along P .

Proof: The edges in a tree with artificial alternate between increasing and decreasing along any path in the tree because the distance from an edge to the root alternates between even and odd numbers. Hence, in step 2, cases (a) and (b), the lemma is certainly true. In cases (c) and (d), if one end of P is not an artificial, then the edges of P

from e_ℓ to that end alternate because the edges were designated increasing and decreasing alternately beginning at e_ℓ . In case (c), e_ℓ is increasing, and in case (d), e_ℓ is decreasing. The proof of the lemma will be completed if it is shown that in case (c) the edge e in P next to e_ℓ and toward a root with artificial is increasing. A vertex of e_ℓ toward a root with artificial is in V_1 in case (c) and is in V_3 in case (d). Hence, in case (c) the vertex of e furthest from the root is in V_1 and in case (d) it is in V_3 . But in a tree with artificial at the root, an increasing edge is an even edge so its vertex furthest from the root is an odd vertex and, hence, in V_3 , and a decreasing edge is an odd edge so its vertex furthest from the root is an even vertex and, hence, in V_1 . Therefore, in case (c) the edge e is decreasing, and in case (d) it is increasing.

Lemma 10 The changes of variables in step 3 does not change $\sum_{j=1}^n a_{1j} x_j$ in a cycle or in P except at the ends of P where the changes are compensated by changes in slacks or artificials.

Proof: Lemma 4 proves the lemma in P except at the ends of P .

For an end of P with a cycle, the change in the variable corresponding to the end edge of P is compensated for by the change in the variables corresponding to two edges of the cycle incident to the end edge.

The change in $\sum_{j=1}^n a_{1j} x_j$ for other vertices v_i of the cycle is zero

by the alternating nature of the edges.

For an end of P with slack, the change in slack was defined so as to compensate for the change in the variable corresponding to the end edge.

For an end of P with y_1 , the end edge is increasing so y_1 decreases.

Lemma 11 In step 2, except for case (c) when $v_1 \in V_1$, $v_j \in V_2$ and case (d) when $v_1 \in V_3$, $v_j \in V_2$, the resulting change in step 3 causes the sum of the artificials to strictly decrease.

Proof: Step 4(ii) assures that increasing edges are less than their upper bounds and decreasing edges are greater than their lower bounds in trees with artificials at the roots. Hence, cases (a) and (b) of step 2 always result in a decrease in the artificial at the root. Cases (c) and (d) for which v_1 and v_j are both in V_1 or both in V_3 , but in different trees of F_B , always result in a decrease in both artificials. The remaining consideration is cases (c) and (d) when v_1 and v_j are both in the same tree and both in V_1 or both in V_3 . In the path P , variables corresponding to increasing edges can increase and variables corresponding to decreasing edges can decrease because they are in a tree with artificial. Let the cycle be $(v_1, e_1, v_2, \dots, v_{2k+1}, e_{2k+1}, v_1)$ where the end of P is incident to v_1 and e_ℓ is the entering edge. Then the edges other than e_ℓ are increasing or decreasing the same as they were before in the tree because e_1 and e_{2k+1} are both opposite of the end edge, and the edges alternate from e_1 and e_{k+1} until e_ℓ is met. In case (c), e_ℓ will be designated as increasing and in case (d) as decreasing just as in the proof of lemma 4. Therefore, the variable corresponding to every increasing edge can increase, and the variable corresponding to every decreasing edge can decrease so the artificial strictly decreases.

Lemma 12 If α , σ , and b are integer, then all variables are

integer except possibly for edges in a cycle which are integer divided by two, and the change θ in step 3 is always integer.

Proof: Initially the lemma is true. Suppose it is true at the beginning of an iteration. Then the permissible change in a variable is an integer unless it is an edge of a cycle in which case the change is an integer divided by two. But the variables all change by θ except for the edges of the cycle which change by $\frac{\theta}{2}$. The size of θ is determined either by $\theta = k$ or $\frac{\theta}{2} = \frac{k}{2}$, so θ is integer. All of the variables change by θ except around a cycle, so they remain integer except around the cycle where they remain integer divided by two.

Theorem 6 At most, m iterations of the algorithm can occur in sequence without any change in the artificials, and the algorithm terminates in a finite number of steps. If α , σ , and b are integer and $\sum_{i=1}^m |b_i| = M$, then the algorithm terminates in at most Mm steps.

Proof: By lemma 6, the only iterations that could result in no decrease in artificials are iterations for which in step 2, cases (c) or (d) occur with one vertex in V_2 . Since $\theta = 0$, the blocking variable could not be in the tree with artificial and is not the entering variable. Hence, the tree with artificial will grow by at least the entering edge. Therefore, V_2 decreases in size by at least one vertex in every iteration for which $\theta = 0$. There are only m vertices so no more than m iterations in sequence could occur with no change in artificials.

Finiteness follows in the same way as for the max-flow algorithm and indeed for linear programs in general once the objective has

been shown to decrease every finite number of iterations. The bound mM when α , σ , and b are integer follows just as before.

This algorithm enjoys the same advantages as the max-flow algorithm and for similar reasons. Here, cycles can form, but the essential fact is that for edge e_k in a tree with artificial, every change in variables in step 3 causes x_k to increase if e_k is an even edge and to decrease if e_k is an odd edge.

The primal-dual method explained for network flows in Chapter I, section 6, applies here in exactly the same way, except that the matrix A is different, and $\pi_i - \pi_j$ is replaced by $\pi_i + \pi_j$ for this problem.

9. Integer Programming in an Undirected Graph

The integer programming problem considered in this section is:

$$(B) \quad \begin{aligned} Ax + Us + Iy &= b, \quad 0 \leq x \leq \alpha, \quad 0 \leq s \leq \sigma, \quad 0 \leq y, \quad x \text{ and } s \\ &\quad \sum_{i=1}^n y_i = w(\min) \quad \text{integer,} \end{aligned}$$

where A and U are the same as in the previous section, and $b \geq 0$, $\alpha > 0$, and $\sigma > 0$ all have integer components.

By lemma 7, the alternating path algorithm gives integer answers to (2) except around odd cycles in the basis. The odd cycles will now be handled so as to avoid non-integer solutions.

The idea of the algorithm is the familiar cutting plane method used by Dantzig, Fulkerson, and Selmar Johnson on the traveling salesman problem [1] and systematically developed by Gomory [5]. This algorithm is similar to one that Edmonds [4] has used to solve a special case of (2), the degree constrained subgraph problem, which is discussed in the next section.

Inequalities of the form

$$(4) \quad \sum_{k \in K} x_k \leq \beta$$

can be thought of as being adjoined to the system. The inequalities (4) are such that every integer solution to (3) satisfies them, but non-integer answers which might arise in the alternating path algorithm do not satisfy them. The following lemma tells exactly the type of inequalities which will be used.

Lemma 13 Let V_0 be a subset of the vertices of G and E_0 be a subset of the edges of G such that every edge of E_0 is incident to at least one vertex of V_0 . Suppose that for all $v_1 \in V_0$ neither s_1^+ nor s_1^- exists. Let $K = \{k \mid e_k \in E_0\}$, $L = \{k \mid e_k \in E_0 \text{ and only one vertex of } e_k \text{ is in } V_0\}$, $M = \{k \mid e_k \in E_0 \text{ and both vertices of } e_k \text{ are in } V_0\}$, and $J = \{i \mid v_1 \in V_0\}$. Suppose $\sum_{i \in J} b_i + \sum_{k \in L} \alpha_k = 2\beta + 1$ where

β is a positive integer. Then, every integer solution to the system (3) satisfies the inequality $\sum_k x_k \leq \beta$.

Proof: Let x, s, y be an integer solution to $Ax + Us + Iy = b$, $0 \leq x \leq \alpha$, $0 \leq s \leq \sigma$, $0 \leq y$. Summing the rows i of $Ax + Us + Iy = b$ for $i \in J$ gives

$$(5) \quad 2 \sum_{k \in M} x_k + \sum_{k \in L} x_k \leq \sum_{i \in J} b_i$$

because there are no slacks in row i for $i \in J$, and the only variables omitted are y_i and x_k , $k \notin K$, and for them $a_{ik} x_k \geq 0$ and $y_i \geq 0$. Adding $\sum_{k \in L} x_k$ to both sides of (5) gives

$$(6) \quad 2 \left(\sum_{k \in M} x_k + \sum_{k \in L} x_k \right) \leq \sum_{i \in J} b_i + \sum_{k \in L} x_k$$

From $x_k \leq \alpha_k$ and $K = L \cup M$ follows

$$(7) \quad 2 \sum_{k \in K} x_k \leq \sum_{i \in J} b_i + \sum_{k \in L} \alpha_k = 2\beta + 1, \quad \text{or}$$

$$(8) \quad \sum_{k \in K} x_k \leq \beta + \frac{1}{2}.$$

But, the left-hand side of (8) is an integer so the right-hand side, $\beta + \frac{1}{2}$, can be lowered to the next smaller integer, β , and the proof is completed.

Since for a given graph G there are only a finite number of inequalities (4) of the type given in lemma 8, the system with them adjoined is still finite. That such inequalities are sufficient to give integer answers to (5) is proven constructively by the algorithm.

Values of x , s , artificials y , and dual variables ρ will be kept track of throughout the algorithm and proven optimal at the conclusion. However, the algorithm differs from the previous ones in that x , y , s may not form a basic solution. The inequalities (4) are not kept track of during the algorithm, but at the conclusion such inequalities are formed to prove optimality. Only the phase I procedure will be done; that is, the problem of minimizing $w = \sum y_i$. The algorithm is similar to the alternating path algorithm of the preceding section.

Vertex-clusters will be used in the algorithm. A vertex-cluster is a set U_i of vertices and other vertex-clusters together with a set E_i of edges. If the vertex-clusters in U_i are thought of as single vertices, then the U_i , E_i form a graph with one cycle.

Such graphs were discussed in section 1. The set V_0 will be a set of certain of the vertices of G included in some vertex-cluster, and E_0 will be a set of certain edges incident to at least one vertex of V_0 .

The vertex-clusters are nested; that is, some of them are included in others. This order of inclusion is important in the variable change step of the algorithm. There, the maximal vertex clusters are first thought of as single vertices in order to determine certain variable changes, and then the variable changes are determined within each vertex-cluster involved beginning with the largest and working down until the vertex-clusters consist only of vertices of G .

If the maximal vertex-clusters are thought of as single vertices, then they, together with vertices and edges, form trees rooted at vertices or vertex-clusters with an artificial. Let F denote the resulting forest. The forest F does not include any vertices, vertex-clusters, or edges within the vertex-clusters, and the maximal vertex-clusters are considered to be vertices of F . As before, the even edges in F are designated as increasing edges and the odd edges in F as decreasing edges.

To begin, set $y_1 = b_1$ if $b_1 > 0$, and the forest F consists of vertices with $b_1 > 0$. There are no vertex-clusters U_k , E_k . V_0 is empty, and E_0 is empty.

Integer Alternating Path Algorithm

Step 1: The dual variables p_1 are zero for vertices not in the forest F of trees with artificials at the root. In the forest F , $p_1 = +1$ for v_1 an even vertex, and $p_1 = -1$ for v_1 an odd vertex. The vertices

v_1 in U_k for all vertex-clusters have $\rho_1 = +1$, and the edge set E_0 has an associated dual variable $\rho_{m+1} = -2$.

Let $V_1 = \{i \mid \rho_i = +1\}$, $V_2 = \{i \mid \rho_i = 0\}$, $V_3 = \{i \mid \rho_i = -1\}$.

Step 2(i): Search for an edge e_ℓ or vertex v_1 satisfying one of the following:

- (a) $e_\ell = [v_1, v_j]$, $x_\ell < \alpha_\ell$, $v_1 \in V_1$, and $v_j \in V_2$;
- (b) $e_\ell = [v_1, v_j]$, $x_\ell > 0$, $v_1 \in V_3$, and $v_j \in V_2$;
- (c) $e_\ell = [v_1, v_j]$, $x_\ell = \alpha_\ell$, $v_1 \in V_0$, and $v_j \in V_2$;
- (d) $v_1 \in V_0$, and s_1^+ or s_1^- exists;
- (e) $v_1 \in V_1$, $v_1 \notin V_0$, and $s_1^+ < \sigma_1^+$ or $s_1^- > 0$;
- (f) $v_1 \in V_3$, and $s_1^+ > 0$ or $s_1^- < \sigma_1^-$;
- (g) $e_\ell = [v_1, v_j]$, $x_\ell < \alpha_\ell$, $e_\ell \notin E_0$, $v_1 \in V_1$, and $v_j \in V_1$;
- (h) $e_\ell = [v_1, v_j]$, $x_\ell > 0$, $v_1 \in V_3$, and $v_j \in V_3$;
- (i) $e_\ell = [v_1, v_j]$, $x_\ell = \alpha_\ell$, $v_1 \in V_0$, and $v_j \in V_3$;
- (j) $e_\ell = [v_1, v_j]$, $0 < x_\ell < \alpha_\ell$, $v_1 \in V_0$, and $v_j \in V_3$.

In cases (a) - (c), go to step 2(ii). In cases (d) - (j), go to step 3.

Step 2(ii): In case (a), if $x_\ell = 0$ or if $v_1 \notin V_0$, then change v_j from V_2 to V_3 , adjoin e_ℓ and v_j to F , and return to step 2(i). If $x_\ell > 0$ and $v_1 \in V_0$, then let U_1, E_1 be the largest vertex-cluster containing v_1 . Let $v_j \in U_1$, $v_j \in V_0$, change v_j from V_2 to V_1 , put e_ℓ in E_1 , and put every edge $e = [v_j, v]$ for $v \in U_1 \cap V_0$ in the edge set E_0 . Return to step 2(i).

In case (b), change v_j from V_2 to V_1 , adjoin e_ℓ and v_j to F , and return to step 2(i).

In case (c), let U_1, E_1 be the largest vertex-cluster containing v_1 . Change v_j from V_2 to V_1 , put v_j in U_1 , e_ℓ in E_1 , and e_ℓ in E_0 . Return to step 2(i).

Step 3: In case (d), let U_1, E_1 be the largest vertex-cluster containing v_1 , let v_1 be the root of the tree in F containing U_1 , and let $P = (v_1, e_1, v_2, \dots, v_{i-1}, e_{i-1}, U_1)$ be the path in F from v_1 to U_1 .

In cases (e) and (f), let v_1 be the root of the tree containing v_1 and let $P = (v_1, e_1, v_2, \dots, v_{i-1}, e_{i-1}, v_i)$ be the path in F from v_1 to v_i .

In cases (g) - (j), suppose v_i and v_j are in different trees of F . Let v_1 be the root of the tree containing v_i , let v_r be the root of the tree containing v_j , let $(v_1, e_1, v_2, \dots, v_{i-1}, e_{i-1}, v_i)$ be the path in F from v_1 to v_i , and let $(v_j, e_j, v_{j+1}, \dots, v_{r-1}, e_{r-1}, v_r)$ be the path in F from v_j to v_r . Let $P = (v_1, e_1, v_2, \dots, v_i, e_\ell, v_j, e_j, v_{j+1}, \dots, v_r)$. The vertices v_1 and v_r have $y_1 > 0$ and $y_r > 0$. In case (g), e_ℓ is increasing, and in cases (h) - (j), e_ℓ is decreasing.

In all of the above cases, a positive integer change θ can be made in the variables corresponding to F , just as in the alternating path algorithm. Here it must also be shown that an integer change $\pm\theta$ can be made alternately within a vertex-cluster included in the path P . Lemma 14 provides that proof. Go to step 4.

In cases (g) - (j), suppose v_i and v_j are in the same tree of F . Let the path in F from v_i to v_j be $(v_i, e_i, v_{i+1}, \dots, v_{j-1}, e_j, v_j)$ and form the cycle $C = (v_i, e_i, v_{i+1}, \dots, v_{j-1}, e_j, v_j, e_\ell)$.

Let the root of the tree containing v_1 and v_j be v_1 and let $P = (v_1, e_1, v_2, \dots, v_{r-1}, e_{r-1}, v_r)$ be the path in F from v_1 to the cycle. If none of v_1, \dots, v_r is a vertex-cluster, if all increasing edges e_k in P have $x_k \leq \alpha_k - 2$, and if all decreasing edges e_k in P have $x_k \geq 2$; then a variable change with $\theta = 2$ can be made just as in the alternating path algorithm except that lemma 9 is needed to show that a positive integer change can be made through any vertex-clusters in the cycle. Repeat the variable change with $\theta = 2$ until such a change would violate the feasibility conditions $0 \leq x \leq \alpha$, $y \geq 0$. If an increasing edge e_k has $x_k = \alpha_k$ or a decreasing edge e_k has $x_k = 0$, then go to step 4.

Otherwise, let v_q be the vertex in P nearest to v_r such that either v_q is a vertex-cluster, e_{q-1} is a decreasing edge with $x_{q-1} = 1$, or e_q is an increasing edge with $x_q = \alpha_q - 1$. A new vertex-cluster U_h, E_h will be formed. Let v_q, \dots, v_r and all of the vertices or vertex-clusters of the cycle C be in U_h and let e_q, \dots, e_{r-1} and all of the edges of the cycle C be in E_h . The vertex v_q is the base of U_h . Let U_h be in F and remove all of the vertices of U_h and edges of E_h from F . If e_q was an increasing edge with $x_q = \alpha_q - 1$, then let V_0 include v_{q+1}, \dots, v_r and all of the vertices of the cycle C . Otherwise, let V_0 include all of the vertices of U_h . Let E_0 include all of the edges in E_h and all of the edges with both vertices in $U_h \cap V_0$. If an edge $e_k = [v_i, v_j]$ of F has $v_i \in U_h, v_j \notin U_h$, and $0 < x_k < \alpha_k$, then adjoin e_k to both E_h and E_0 and adjoin the vertex v_j to both U_h and V_0 . Remove e_k and v_j from F . Enlarge E_0 to include all edges $e = [v_j, v]$

for v some vertex in $U_h \cap V_0$. If an edge $e_k = [v_1, v_j]$ has $v_1 \in U_h$, $v_j \notin U_h$, and $x_k = \alpha_k$, then adjoin e_k to both E_h and E_0 , adjoin the vertex v_j to only U_h , and remove e_k , v_j from F .

Return to step 1.

Step 4: For $P = (v_1, e_1, v_2, \dots, v_{r-1}, e_{r-1}, v_r)$, if v_1 has an artificial, then let v_q be the vertex in P nearest to v_1 such that either v_q is a vertex-cluster, e_{q-1} is a blocking edge, e_{q-1} is an entering edge, or $v_q = v_r$. Then drop all of P from e_{q-1} to the entering variable from F , and drop from F all of the vertices and edges whose path to the root includes vertices already dropped from F . Delete from V_0 all those vertices and change them from V_1 or V_3 to V_2 . Drop all of the vertex-clusters and drop from E_0 any edge incident to a vertex dropped.

Return to step 1.

Lemma 14 Let U_1 be a vertex-cluster and $v \in U_1$. Then, there is an alternating path P from v to the base of U_1 , and the end edge in P incident to v is a decreasing edge. If $v \in V_0$ as well, then there is also an alternating path with an increasing end edge at v . A positive integer change in variables can be made along the alternating paths.

Proof: The proof is by induction because it is assumed that the lemma is true for every vertex-cluster used in forming the vertex-cluster U_k . With that inductive hypothesis, we can assume all of the vertex-clusters used to form U_1 are single vertices.

Initially, the vertex-cluster U_1 is formed from a path

$P = (v_1, e_1, v_2, \dots, v_{r-1}, e_{r-1}, v_r)$ and a cycle $C = (v_r, e_r, v_{r+1}, \dots, v_q, e_q, v_r)$ where the path P may consist only of the vertex v_r . The cycle C is an odd cycle and has the same alternating character as in the alternating path algorithm.

For any vertex $v_i \in C$, $i \geq r+1$, $P_1 = (v_i, e_i, v_{i+1}, \dots, v_q, e_q, v_r, e_{r-1}, v_{r-1}, \dots, v_2, e_1, v_1)$ and $P_2 = (v_i, e_{i-1}, v_{i-1}, \dots, v_{r+1}, e_r, v_r, e_{r-1}, v_{r-1}, \dots, v_2, e_1, v_1)$ are alternating paths from v_i to v_1 , and one of e_i, e_{i-1} is increasing while the other is decreasing. The paths P_1 and P_2 are simple paths so obviously an integer change can be made.

For any vertex $v_i \in P$, $i \geq 2$, $P_1 = (v_i, e_{i-1}, v_{i-1}, \dots, v_2, e_1, v_1)$ and $P_2 = (v_i, e_i, v_{i+1}, \dots, v_{r-1}, e_{r-1}, v_r, e_r, v_{r+1}, \dots, v_q, e_q, v_r, e_{r-1}, v_{r-1}, \dots, v_2, e_1, v_1)$ are alternating paths from v_i to v_1 , and one of e_i, e_{i-1} is increasing while the other is decreasing. The path P_1 is simple, but P_2 is not simple. But, the increasing edges e_k of P , $k \geq 2$, have $x_k \leq \alpha_k - 2$, and the decreasing edges e_k of P have $x_k \geq 2$. Hence, an integer change can still be made along P_1 and P_2 .

For v_1 , the path $P_1 = (v_1)$ has the effect of having the end edge an increasing edge because at v_1 there is either an artificial or a decreasing edge incident to v_1 on the path leading from v_1 to the root. If $v_1 \in V_0$, then all of the increasing edges e_k of P have $x_k \leq \alpha_k - 2$, and decreasing edges e_k of P have $x_k \geq 2$. Hence, $P_2 = (v_1, e_1, v_2, \dots, v_{r-1}, e_{r-1}, v_r, e_r, v_{r+1}, \dots, v_q, e_q, v_r, e_{r-1}, v_{r-1}, \dots, v_2, e_1, v_1)$ is an alternating path from v_1 to v_1 with e_1 decreasing and permitting an integer change in variables. The proof is complete.

The change of variables in step 3 is now complete. Finiteness will now be proven.

Theorem 7 If $M = \sum_{i=1}^m b_i$, then the algorithm terminates in at most $2 m M$ iterations.

Proof: Every change of variable in step 3 results in an integer decrease in $\sum_{i=1}^m y_i$. Hence, the proof will be completed if it is shown that there can be, at most, $2 m$ iterations in sequence with no change in flow.

If the algorithm goes to step 2(11), then V_2 decreases by one vertex, and V_0 either remains the same or increases. If the algorithm goes to step 3 and no change in variables results, then a new vertex-cluster is formed, and V_0 increases by at least one vertex while V_2 remains the same. There are only m vertices in all, so only $2 m$ such iterations could occur in sequence.

Theorem 8 At the termination of the algorithm, let J , K , L , and M be as in lemma 13 and let q be the number of vertex-clusters in F ; that is, the number of maximal vertex-clusters. Then,

$$(9) \quad \sum_{i \in J} b_i + \sum_{k \in L} \alpha_k = 2\beta + q$$

where β is a positive integer, every integer solution to (3) satisfies

$\sum_{k \in K} x_k \leq \beta$, and the present solution x, s, y is optimal to the

linear program:

$$(10) \quad Ax + Us + Iy = b, \quad 0 \leq x \leq \alpha, \quad 0 \leq s \leq \sigma, \quad 0 \leq y,$$

$$\sum_{k \in K} x_k \leq \beta$$

$$\sum_{i=1}^m y_i = w(\min).$$

Proof: Equation (9) and the fact that every integer solution to (3) satisfies $\sum_{k \in K} x_k \leq \beta$ will be shown together. Suppose that the

maximal vertex-clusters are $(U_1, E_1), \dots, (U_q, E_q)$. Let $V_0^\ell = V_0 \cap U_\ell$ and $E_0^\ell = E_0 \cap E_\ell$. Let $K^\ell = \{k \mid e_k \in E_0^\ell\}$, $L^\ell = \{k \mid e_k \in E_0^\ell \text{ and only one vertex if } e_k \text{ is in } V_0^\ell\}$, $M^\ell = \{k \mid e_k \in E_0^\ell \text{ and both vertices of } e_k \text{ are in } V_0^\ell\}$, and $J^\ell = \{i \mid v_i \in V_0^\ell\}$.

For each $\ell = 1, \dots, q$,

$$(11) \quad 2 \sum_{k \in K^\ell} x_k = \sum_{i \in J^\ell} b_i + \sum_{k \in L^\ell} \alpha_k - 1$$

because at the base of each vertex cluster either there is an edge e_k , $k \in L^\ell$, with $x_k = \alpha_k - 1$, or there is an edge e_k , $k \notin K^\ell$, with $x_k = 1$. All other edges e_k , $k \notin K^\ell$, incident to vertices of V_0^ℓ have $x_k = 0$. There are no slacks on $v_i \in V_0$ because of step 2(i) case (d). Hence, $\sum_{i \in J^\ell} b_i + \sum_{k \in L^\ell} \alpha_k$ is an odd number; say, $2\beta^\ell + 1$.

Then, lemma 13 asserts that every integer solution to (3) satisfies

$$(12) \quad \sum_{k \in K^\ell} x_k \leq \beta.$$

The vertex sets V_0^1, \dots, V_0^q are pair-wise disjoint, and the edge sets E_0^1, \dots, E_0^q are pair-wise disjoint. Summing the equations

$$\sum_{i \in J^\ell} b_i + \sum_{k \in L^\ell} \alpha_k = 2\beta^\ell + 1 \quad \text{for } \ell=1, \dots, q \quad \text{gives} \quad \sum_{i \in J} b_i + \sum_{k \in L} \alpha_k = 2\beta + q,$$

where $\beta = \sum_{\ell=1}^q \beta^\ell$.

Thus, equation (9) is proven, and summing the inequalities (12)

for $\ell = 1, \dots, q$, gives $\sum_{k \in K} x_k \leq \beta$ for every integer solution to (3).

To prove optimality, the complimentary slackness conditions (page 134, 23) will be used since the solution is no longer basic. The dual variables are $\rho_i = +1$ for $v_i \in V_1$, $\rho_i = 0$ for $v_i \in V_2$, $\rho_i = -1$ for $v_i \in V_3$, and $\rho_{m+1} = -2$. The following conditions together with complimentary slackness prove optimality:

- (13) if $e_k \in E_0$, $e_k = [v_i, v_j]$, then $\rho_i + \rho_j + \rho_{m+1} = 0$;
- (14) if $e_k \notin E_0$, $e_k = [v_i, v_j]$, $x_k = \alpha_k$, then $\rho_i + \rho_j \geq 0$;
- (15) if $e_k \notin E_0$, $e_k = [v_i, v_j]$, $0 < x_k < \alpha_k$, then $\rho_i + \rho_j = 0$;
- (16) if $e_k \notin E_0$, $e_k = [v_i, v_j]$, $x_k = 0$, then $\rho_i + \rho_j \leq 0$;
- (17) if $s_i^+ = \sigma_i^+$ or $s_i^- = 0$, then $\rho_i \geq 0$;
- (18) if $0 < s_i^+ < \sigma_i^+$ or $0 < s_i^- < \sigma_i^-$, then $\rho_i = 0$.
- (19) if $s_i^+ = 0$ or $s_i^- = \sigma_i^-$, then $\rho_i \leq 0$.
- (20) $\sum_{k \in K} x_k = \beta$.

Condition (13) follows from the observation that if $e_k \in E_0$, then $v_i \in V_1$ and $v_j \in V_1$.

If (14) were violated, then either $v_i \in V_3$ and $v_j \in V_2$, or $v_i \in V_3$ and $v_j \in V_3$. But the algorithm has terminated, and step 2(1) case (b) excludes $v_i \in V_3$ and $v_j \in V_2$, and case (h) excludes $v_i \in V_3$ and $v_j \in V_3$.

Similarly for (15), step 2(1) case (a) excludes $v_i \in V_1$ and $v_j \in V_2$, case (b) excludes $v_i \in V_3$ and $v_j \in V_2$, case (g) excludes $v_i \in V_1$ and $v_j \in V_1$, and case (h) excludes $v_i \in V_3$ and $v_j \in V_3$. Hence, either $\rho_i \in V_1$ and $\rho_j \in V_3$, or $\rho_i \in V_2$ and $\rho_j \in V_2$. In either case, $\rho_i + \rho_j = 0$.

For (16), step 2(1) case (a) excludes $v_i \in V_1$ and $v_j \in V_2$, and case (g) excludes $v_i \in V_1$ and $v_j \in V_1$. Hence, $\rho_i + \rho_j \leq 0$.

In (17), step 2(1) case (f) assures that $v_1 \notin V_3$ so $\rho_1 \geq 0$.

In (18), step 2(1) cases (e), and (f) assure that $v_1 \in V_2$ and $\rho_1 = 0$.

In (19), step 2(1) cases (d) and (e) assure that $v_1 \notin V_1$ and $\rho_1 \leq 0$.

Equation (11) proves that $\sum_{k \in K} x_k = \beta^\ell$, and summing for $\ell = 1, \dots, q$ gives (20).

Corollary 1 If G has no odd cycles or if every odd cycle in G has at least one vertex with a slack permitted, then no vertex clusters need be formed and no inequalities need be adjoined to the linear program (5) in order to find an integer answer.

Proof: The proof follows from the fact that V_0 has no slacks at any of its vertices.

Corollary 2 An integer solution x, s, y to (5) is optimal if, and only if, there does not exist an alternating path P in G with an artificial $y_1 > 0$ at one end and an increasing edge at that end of P , and the other end of P having a slack or artificial which can change to compensate for the change in the other end edge of P . The path P need not be simple.

10. The Degree-Constrained Subgraph Problem

The degree $\rho(v)$ of a vertex v of a graph G is the number of edges incident to v . Let H be a subgraph of G and let the degree of a vertex v in H be denoted $\rho'(v)$. The integer program in an undirected graph of the preceding section can be interpreted as the following problem: if an edge e_k can be repeated α_k times in determining the degrees $\rho'(v)$ of vertices v in H , then find, if

possible, a subgraph H of G such that $b_1 \leq \rho'(v_1) \leq b_1 + \sigma_1^-$ if s_1^- exists, $b_1 - \sigma_1^+ \leq \rho'(v_1) \leq b_1$ if s_1^+ exists, and $b_1 = \rho'(v_1)$ if no slack is permitted at v_1 . Here, if $\sigma_1^- = +\infty$, then there is no upper bound on $\rho'(v_1)$, and if $\sigma_1^+ = +\infty$, then there is no lower bound on $\rho'(v_1)$, although zero is always an implied lower bound on v_1 .

This problem has been studied by Berge [1], Norman and Rabin [9], and Edmonds [4] and [5].

Corollary 3 (Berge, Norman, Rabin) Among all subgraphs H of G having $\rho'(v_1) \leq b_1$, a given subgraph H_0 has the maximum number of edges if, and only if, there is no alternating path between two vertices v_1 and v_2 of H_0 such that $\rho'(v_1) < b_1$, $\rho'(v_2) < b_2$, and the alternating path has increasing edges at each end.

Proof: This corollary follows from corollary 2 applied to the following special case of (2):

$$(21) \quad Ax + Iy = b, \quad 0 \leq x \leq \alpha_k, \quad 0 \leq y, \quad x \text{ integer}$$

$$\sum_{i=1}^m y_i = w(\min).$$

Subtracting the rows of $Ax + Iy = b$ from the objective $\sum_{i=1}^m y_i$ and dividing by -2 converts (21) into

$$(22) \quad Ax + Iy = b, \quad 0 \leq x \leq \alpha_k, \quad 0 \leq y, \quad x \text{ integer.}$$

$$\sum_{k=1}^n x_k = z(\max).$$

REFERENCES

1. Berge, C., "Two Theorems in Graph Theory," Proc. Nat. Acad. Sci., 43(1957), 842.
2. Dantzig, G.B., Linear Programming and Extensions, Princeton University Press, Princeton, New Jersey, 1963.
3. Dantzig, G.B., L.R. Ford, Jr., and D.R. Fulkerson, "A Primal-Dual Algorithm for Linear Programs," Linear Inequalities and Related Systems, Annals of Mathematics Study Number 38, Princeton University Press, Princeton, New Jersey, 1956, 171-481.
4. Edmonds, J. "Covers and Packings in a Family of Sets," Bull. Amer. Math. Soc., 68(1962), 494-499.
5. Edmonds, J., "Paths, Trees, and Flowers," National Bureau of Standards, Washington, D.C., 1962.
6. Ford, L.R., Jr. and D.R. Fulkerson, "Solving the Transportation Problems," Management Science, 3(1956), 24-32.
7. Ford, L.R., Jr. and D.R. Fulkerson, Flows in Networks, Princeton University Press, Princeton, New Jersey, 1963.
8. Jewel, W.S., "Optimal Flow through Networks with Gains," Operations Research, 10(1962), 476-499.
9. Norman, R.Z. and M.O. Rabin, "An Algorithm for a Minimum Cover of a Graph," Notices of the Amer. Math. Soc., 5(1958), 36.
10. Scoins, H.I., "The Compact Representation of a Rooted Tree and the Transportation Problem," International Symposium on Mathematical Programming, London, 1964.